




Instituto Tecnológico de Colima



Informe técnico de residencia profesional

Departamento de ingeniería eléctrica y electrónica

Ingeniería mecatrónica

Asesor:

Armando Gaytán Godínez

Sistema de visión computacional para detección y clasificación de jitomates adaptable a un robot de cosecha.

Dorian Alexis Velázquez Sotomayor

14460444






Villa de Álvarez, Col., a 15 de Julio de 2019

Indice

Indice de figuras

Indice de tablas

Introducción

El instituto tecnológico de Colima se encuentra ubicada en la avenida Tecnológico No. 1 en Villa de Álvarez, Colima; Es una escuela homologada al sistema de los Tecnológicos Nacionales de México, que ofrece diversas carreras en ingeniería y licenciaturas.

El proyecto busca desarrollar un problema concerniente a robots cosechadores que pueda ser resuelto utilizando visión computacional, seleccionando un producto en especifico que sea de la región y sea de fácil acceso, eligiendo resolver lo relacionado a la detección de jitomates maduros en un invernadero.

Se realiza una investigación a modo del estado del arte para conocer los principales exponentes de los robots cosechadores y ver soluciones posibles al problema, así como una investigación documental de las soluciones posibles y enfoques usados para resolver un problema de segmentación y detección, con soluciones tanto con procesamiento de imagen como con el uso de redes neuronales.

En el presente documento se desarrollan diversas técnicas novedosas utilizadas en redes neuronales convolucionales, presentando un modelo mejorado que tiene ventajas sobre el original en cuestión de entrenamiento.

Justificación

Debido a problemas mundiales como la gran demanda alimenticia, cada vez se vuelve más complicado resolver los problemas de producción masiva de productos agrícolas, esto sumado con los problemas de migración hacia las ciudades y con el escaseo de la mano de obra en el campo, hace indispensable pensar en soluciones novedosas a las técnicas habituales de agricultura y producción.

Uno de los costos principales dentro de la producción de productos agrícolas es el que concierne a la cosecha, llegando a representar hasta un 30% del costo total de producción *(Juste et al., 1991; Dale Whittaker,*

G. E. Miles, O. R. Mitchell, & L. D. Gaultney, 1987) , además de que la mano de obra suele ser exclusivo de temporada, que imposibilita la adquisición del producto todo el tiempo del año.

La introducción de los robots cosechadores en la automatización de procesos agrícolas permite la apertura de nuevos mercados de comercio, ya que mejora la calidad del producto final.

Al considerar la automatización de los robots cosechadores, se plantea una de los principales problemas a tratar, la visibilidad de la fruta *(D. M. Bulanon, T. F. Burks, & V. Alchanatis, 2009)*, principalmente por las características del entorno y las limitaciones de la visión de los robots cosechadores (90% frente a 85% de los sistemas de visión computacional *(Juste et al., 1991; Bac, van Henten, Hemming, & Edan, 2014)*), cuestión que impacta notablemente en el desempeño.

Por ello se considera relevante el estudio de la detección de frutas en robots de cosecha para la mejora de los procesos de visión de un robot cosechador y subsecuentemente mejorar la producción agrícola.

Como profesionista, el aprender sobre visión computacional permite generar panoramas para solucionar problemas con características principalmente visuales, además el aprender al respecto de inteligencia artificial y redes neuronales abre campos de aplicación nuevos para solucionar problemas robustos y más complejos, produciendo soluciones más robustas y con menos pasos, como lo es la solución de problemas con técnicas de procesamiento de imágenes.

Objetivos

Objetivos generales

- Investigar y desarrollar técnicas de visión computacional para la detección y clasificación de jitomates maduros en un invernadero
- Plantear las consideraciones pertinentes para la adaptación a un robot mecánico de cosecha.

Objetivos específicos

- Generar un sistema que sea invariante a condiciones ambientales de iluminación, condiciones de contraste, etc;
- Generar un sistema que sea invariante a la oclusión de frutas.
- Lograr una detección correcta de frutas de más allá del promedio global (85%).
- Desarrollar una técnica basada en redes neuronales convolucionales.
- Modificar alguna arquitectura neuronal para obtener alguna mejora en características de entrenamiento y clasificación.

Problemas a resolver

Se requiere conocer el estado del arte de robots cosechadores y los distintas técnicas existentes para poder formular, diseñar y analizar distintas soluciones a distintos productos, frutas y verduras.

Investigar las características de la solución del problema, limitaciones y ventajas físicas y económicas, debido al hardware a utilizar, tanto en el desarrollo como en la producción.

Elegir un producto para la implementación del problema, teniendo que ser de la región y accesible (se decidió jitomates rojos, debido a la facilidad de implementación y a lo accesible del producto).

Indagar en el estado del arte de las soluciones a problema de detección utilizando inteligencia artificial, en específico redes neuronales convolucionales, así como consideraciones de mejora en redes neuronales.

Proponer una solución al problema de detección de jitomates utilizando redes neuronales convolucionales, modificando o agregando partes que logren un mejor desempeño de la red.

Investigar y en caso de no existir generar un dataset de imágenes etiquetadas de jitomates que estén en relación con el tipo de solución propuesta.

Desarrollar distintas simulaciones que permitan probar y subsecuentemente depurar el sistema, llegando a conclusiones sobre la propuesta elegida.

Procedimiento y descripción de las actividades realizadas

Se realizó una investigación sobre los aspectos básicos de los robots cosechadores, así como una cronología del estado del arte desde su desarrollo hasta la actualidad, dándole énfasis a la parte de visión computacional, consultando diversos artículos y reviews del estado del arte enfocados a diversos productos agrícolas, frutos y vegetales.

Fue necesario generar una investigación sobre los aspectos de visión computacional y su uso con redes neuronales en apoyo a los robots cosechadores, desde sus primeras apariciones hasta la actualidad.

Se efectuó una investigación sobre las redes neuronales convolucionales y su uso en tareas de visión computacional, detección, clasificación, etc; las partes principales que la conforman, así como algunas arquitecturas ya creadas.

Se ejecutaron pruebas sobre problemas ya resueltos de detección, clasificación y segmentación para la familiarización de la tecnología para el desarrollo de redes neuronales convolucionales y la plataforma de desarrollo.

Se realizó la búsqueda, descarga, discriminación y etiquetado de imágenes de jitomates para la creación de un dataset de segmentación de plantas de jitomates utilizando la herramienta de etiquetado labelme, que permite exportar en un formato compatible con otras implementaciones de redes neuronales, permitiendo en un futuro el reuso de los datos de entrenamiento para otros fines.

Se programó una red neuronal basada en la arquitectura U-Net con modificaciones para trabajar en canales a color y se realizaron pruebas de entrenamiento con los datos creados, utilizando la plataforma de trabajo jupiter en el servidor google colab.

Estado del arte de robots cosechadores.

El diseño de los prototipos robóticos usualmente consiste en 3 unidades principales, la primera unidad es un sistema de reconocimiento que identifica y localiza la fruta, la segunda unidad es el sistema de toma que realiza operaciones de toma y corte; la tercera parte es un sistema de movimiento programado en base a la geometría del espacio en el que operaría y la estructura del robot (Figura 1).

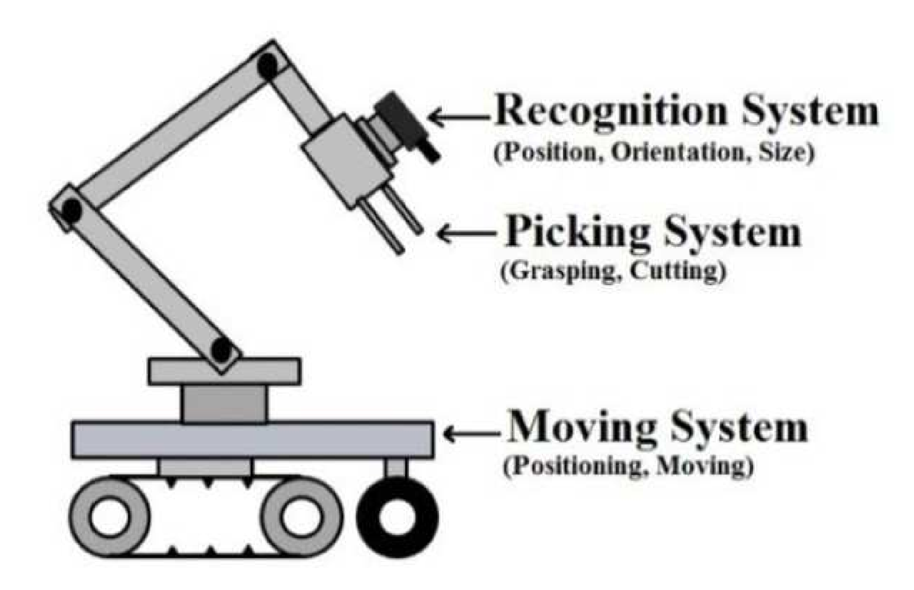


Figura 1: Modelo de robot cosechador (Bachche, 2015)

Dependiendo de la aplicación agrícola y del espacio en que el robot operaría, se consideran juntas rotacionales, juntas lineares, juntas ortogonales, etc,
actuadores hidráulicos, neumáticos, actuadores lineales, motores eléctricos, también mecanismos como engranes, bandas, o uniones con los actuadores, así como sensores como termocuplas, cámaras, encoders ópticos, etc;

El control de movimiento puede ser realizado de diversas maneras, como control de velocidad, control de posición, control de fuerza o presión, control electrónico de motores etc, (Bachche, 2015).

Existen problemas que vuelven complicada la implementación de soluciones robóticas en la automatización de la agricultura, algunos de estos problemas son:

- El futuro de la agricultura.
- Bajo valor del producto.
- Espacio de trabajo variable.
- Énfasis de los desarrolladores.

El futuro de la agricultura no está muy claro, no hay políticas certeras y muchos mercados son inestables o han desaparecidos, por otro lado la presión de la competencia resultará en el uso de robots y máquinas inteligentes que sean capaz de hacer cargo.

Se requiere de una tecnología capaz de ser viable económicamente para los precios de los cultivos actuales.

la posición de las plantas y orientación en las hileras pueden variar de planta a planta, los campos tienen formas irregulares y topografía accidentada, un robot debe ser capaz de trabajar bajo estas condiciones.

Las consideraciones del entorno son importantes para el desarrollo del robot, dependiendo del espacio y de la capacidad de producción, puede ser una huerta, un invernadero, plantaciones en interiores y plantaciones en campo abierto (Figura 2), esto determina los problemas a los que hay que enfrentarse en cuestiones mecánicas y de desarrollo.

Las variaciones del entorno más habituales suelen ser: vientos y lluvias, luz cambiante, cambios en la visibilidad del objeto, cambios en la accesibilidad del objeto, facilidad de manejo y facilidad de instalación del robot. Teniendo menores variaciones en cultivos de interiores y mayores variaciones los cultivos en exteriores (Bachche, 2015).



Figura 2: Entornos de trabajo de robots cosechadores, espacios abiertos, invernaderos, interiores y huertas respectivamente (Bac, van Henten, Hemming, & Edan, 2014).

Aún falta mucho para lograr las capacidades de un humano y los robots necesitarán mucho tiempo para lograr realizar más de una tarea específica (Sistler, 1987).

Se presenta a continuación una cronología de los principales robots cosechadores, mostrando características relevantes de manera cronológica, presentando desde sus inicios hasta la actualidad, mostrando el avance que se ha tenido y la diversidad de productos desarrollados desde sus inicios hasta la fecha (Figura 3).

El estudio de los robots cosechadores en la producción comenzó en 1968 (C. E. Schertz and G. K. Brown, 1968), no se llegó a gestionar ningún robot físico sino hasta el año de 1984, (Kawamura et al,1984) cuando se desarrolló el primer robot cosechador en La Universidad de Kyoto, diseñado para cosechar tomates, el robot consistía en un robot de 5 grados de libertad con visión estereoscópica.

Se incorpora en 1985, España y Francia creando el proyecto MAGALi (d'Esnon, 1985) enfocado en cosechar manzanas.

En 1987, Sistler (Sistler, 1987) hace una reseña sobre máquinas inteligentes y la posibilidad en la agricultura práctica, presentando los retos a enfrentarse y el posible futuro de la agricultura asistida por máquinas inteligentes.

En 1989 Amaha (Amaha et, 1989) desarrolla un robot cosechador de pepino en Tokio, Japón, siendo el primer prototipo generado en dicho país.

Whitney y Harrell desarrollaron un brazo robótico que tenía una velocidad de recolección de una fruta cada 5 segundos (Whitney & Harrell, 1989), enfocándose al diseño mecánico.

Sevilla en 1989 (Sevilla et al,1989) desarrolla un estudio sobre un robot cosechador de uvas.

En 1990 Tilltet (Tilltet, 1990) reporta un cosechador mecatrónico en Reino Unido que consistía en un robot cartesiano y un sistema de visión blanco y negro.

Sandini en cambio (Sandini, Buemi, Massa, & Zucchini, 1990) desarrolla en ese mismo año, un robot cuyas operaciones eran navegar a través de un invernadero y localizar jitomates, utilizando dos cámaras PAL a color.

Harrell, Adsit, Pool y Hoffman (R. C. Harrell, P. D. Adsit, T. A. Pool, & R. Hoffman, 1990) desarrollan un robot móvil para el estudio de condiciones en la cosecha de cítricos, utilizando múltiples sensores y actuadores.

Kubota Co. (Hayashi, 1991) en Japón desarrolla un robot de 4 gdl para cosechar naranjas, utilizando un sistema óptimo de proximidad en el gripper con luz estroboscópica con cámaras en el manipulador.

Pool y Harrell (T. A. Pool & R. C. Harrell, 1991) reportaron el desarrollo de un prototipo para cosechar naranjas con labios giratorios para tomar las frutas de manera suave.

Kassay (Kassay,1992) elabora un robot para cosechar manzanas, consiste de 2 cámaras de color con detección de frutas y posicionamiento de frutas usando 6 brazos, visión estereoscópica y 4 dedos.

(Israel, Benady y Miles, 1992) desarrollan un robot cartesiano cosechador de melones, acoplado a un tractor utilizando un proyector lineal láser basado en la curvatura para la detección de melones.

Tillet (Tillett, 1993) desarrolla un robot manipulador usado para manejar material biológico en horticultura.

Reed y Tillett (Reed & Tillett, 1994) han desarrollado un sistema de localizar y recolección de hongos, utilizando un sistema de visión en blanco y negro.

Grattolli (Grattoni, Cumani, Guiducci, & Pettiti, 1994) reporta un equipo móvil con visión estereoscópica para espárragos.

En España se desarrolla Agribot (Buemi, 1995), un robot asistido para el corte de frutas.

Edan desarrolla varios sistemas para la guía autónoma de robots dentro de invernaderos (Edan, 1995).

Kondo, Monta y Fujiura reportaron varios robots cosechadores, de tomates y pepinos, con sistema robótico de movimiento y navegación (Kondo, Monta, & Fujiura, 1996).

Kondo y Fujiura desarrollaron un sistema robótico que consistía en un manipulador articulado de 6 gdl, cámara monocromática con filtros ópticos y un detector de pedúnculos, discriminados por características morfológicas.
(Kondo & Fujiura, 1996).

En España desarrollan AURORA , un movil autónomo para diversas tareas dentro de invernaderos (Mandow et al., 1996).

En 1997 (Arndt, Rudziejewski, & Stewart, 1997) desarrollan un sistema robótico para el CAMIA con tiempos bajos de trabajo y un 94% de eficiencia en cosecha.

Kondo y Monta (Kondo & Monta, 1999) desarrollaron dos tipos de robots cosechadores de fresas, tanto para hidroponias como para plantas en tierra, el robot consiste de un manipulador de 3 gdl con manipulador neumático por aspiración y corte con giro y sensores visuales.

En el año 2000 (Scarfe, Flemmer, Bakker, & Flemmer, 2000) desarrollan un sistema de robot autónomo para la recolección de kiwi basado en visión computacional y el uso de comandos por radio, utilizando 4 brazos de recolección y protegiendo el fruto de la lluvia, recolectando una fruta por segundo.

En 2001, (Reed, Miles, Butler, Baldwin, & Noble, 2001) reporta un robot capaz de detectar hongos, escogerlos por tamaño, seleccionando y cortando utilizando un efector final con 5 navajas de alta velocidad y gripper neumático, el sistema de visión es circular con luz fluorescente.

En el 2002 (Brown,2002) desarrolla un análisis de 8 tipos de cortes.

En ese mismo año (van Henten et al., 2002) desarrolla un robot autónomo para la cosecha de pepinos, este robot consistía en un manipulador de 7 gdl, vehiculo autónomo, efector final y sistema de imagen tridimensional para la detección de frutas, con un 95% de éxito.

En el 2002 (Hayashi, Ganno, Ishii, & Tanaka, 2002) desarrollan un sistema robótico para cosecha de berenjenas combinando operaciones de segmentación con colores y control difuso para el manipulador y el sistema de corte de pedúnculo.

En ese mismo año (Cho, 2002) desarrolla un robot cosechador de lechuga con 3gdl utilizando control difuso.

En el 2003 (Van Henten et al., 2003) desarrollan un robot cosechador de pepinos con condiciones ambientales variables.

En el siguiente año (Seiichi Arima, Naoshi Kondo, & Mitsuji Monta, 2004) desarrolla un robot cartesiano de 4 gdl utilizado para la cosecha de fresas.

En el 2005 (Burks et al., 2005) describe los retos para la detección y cosecha robótica de cítricos, con énfasis en sistemas visuales, ofrece además literatura y síntesis sobre problemas a los que se enfrentarían los científicos y horticultores al realizar un sistema óptimo máquina-planta.

En ese mismo año (Sanders, 2005) realiza una investigación detallada de varios aspectos de cosecha que involucran selección, remoción y arreglo en huertos de naranja.

En el 2005 (Kitamura & Oka, 2005) desarrollan un robot cosechador de pimientos dulces basado en rieles y prototipos de engranes. El sistema de visión consistía en un sistema de luz artificial, binarización HSI para el

procesamiento de imágen y sistema de recolección consistente en dedos mecánicos y podadora.

En el 2006 (Foglia & Reina, 2006) desarrollan un sistema de cosecha de radicchio que consiste en un sistema de manipulador doble de 4 barras, un gripper especial y sistema de visión computacional basado en filtrado inteligente de color y operaciones morfológicas para localizar plantas en el campo, el sistema utilizó un sistema de evalucación de desempeño.

En ese mismo año (Belforte, Deboli, Gay, Piccarolo, & Ricauda Aimonino, 2006) desarrollan un sistema robótico multitarea en un invernadero, como tareas de fertilización y rociado.

En el 2007 (Naoshi Kondo et al., 2007) desarrollan un sistema controlador de efector final para la cosecha de racimos de jitomates, fue utilizado un sistema para recolectar el racimo sin generar vibraciones.

En el 2008 (Tanigaki, Fujiura, Akase, & Imagawa, 2008) desarrollan un sistema cosechador de cerezas que consistía en un sistema de 4gdl, un sistema tridimensional de visión, el sistema de visión consistía en un sistema láser de visión y de diodos rojos que escaneaban el objeto de manera simultánea.

En ese mismo año (Baeten, Donné, Boedrij, Beckers, & Claesen, 2008) desarrollan un sistema de recolección de manzanas basadas en un sistema de 6 gdl y un gripper con embudo de silicón con una cámara montada, con éxito del 80% y 8-10s para tomar una manzana.

En el 2009 (Irie, Taguchi, Horie, & Ishimatsu, 2009) desarrollan un sistema de recolección de espárragos combinando visión tridimensional y sistema robótico telescópico.

En ese mismo año (Van Henten, Van't Slot, Hol, & Van Willigenburg, 2009) desarrollan una investigación sobre el sistema robótico óptimo para la cosecha de pepino, encontrando que es efectivo un robot con manipulador PPRR de 4 eslabones.

En 2009 (Chatzimichali, Georgilas, & Tourassis, 2009) desarrollan un sistema para moverse a través de un campo de espárragos blancos, identificarlos, tomarlos y cortarlos sin generar daño físico.

En 2010 (Aljanobi, Al-hamed, & Al-Suhaibani, 2010) desarrollan un manipulador industrial de 6 gdl para recolección de dátiles.

En ese mismo año (Hayashi et al., 2010) desarrollan un sistema de cosecha de fresas usando un manipulador cilíndrico, un efector de succión, sistema de movimiento y de visión, con éxito del 60%.

En 2011 (De-An, Jidong, Wei, Ying, & Yu, 2011) presentan un sistema de cosecha de manzanas usando una estructura PRRRP basadas en manipulador en forma de cuchara, gripper con actuador neumático y servocontrol de sistema basada en visión usando máquinas de vectores de soporte, con éxito del 77% y un tiempo de 15 segundos por manzana.

En 2011 (Kohan ,2011 desarrolla un sistema de recolección de rosas usando un sistema de visión estereoscópica y un sistema de manipulación de 4 gdl, con éxito del 82.22%.

En ese mismo año (Li, Lee, & Hsu, 2011) investiga distintos métodos de recolección de frutas para robots, categorizados de acuerdo a sus métodos de visión y análisis de imágenes.

En el 2012 (Feng, Q., Wang, X., et al, 2012) menciona un sistema de recolección de fresas usando cámaras con sensores sonares y un sistema de navegación autónomo, fue utilizado un sistema de 6gdl con 86% de éxito.

En ese mismo año (Li & Kongzhi-Lilun-Zhuanye-Weiyuanhui, 2012) desarrolló el diseño y simulación de un sistema de cosecha de tomates usando un manipulador de 4gdl.

En el 2013 (Zhenyu Yang et al., 2013) desarrolla un sistema de cosecha de pepinos basado en un monoriel con sensores infrarojos y cámaras, usando técnicas de visión por transformación de grises, detección de bordes y algoritmos de umbralados por varianza de máximos locales, el éxito de cosecha fue del 97% a 0.6m/seg.

En el año siguiente (Hemming, J et al, 2014) reportan un robot con 9 gdl usado para la cosecha de pimientos dulces, dos cámaras de 5 megapixeles con sistema de visión tridimensional e iluminación artificial.

En el 2014 (Bac, van Henten, Hemming, & Edan, 2014) analizan el estado del arte de los sistemas y provee perspectivas para la cosecha de cultivos, enfatizando los entornos de producción, con distintas perspectivas, indicadores de rendimiento, etc.

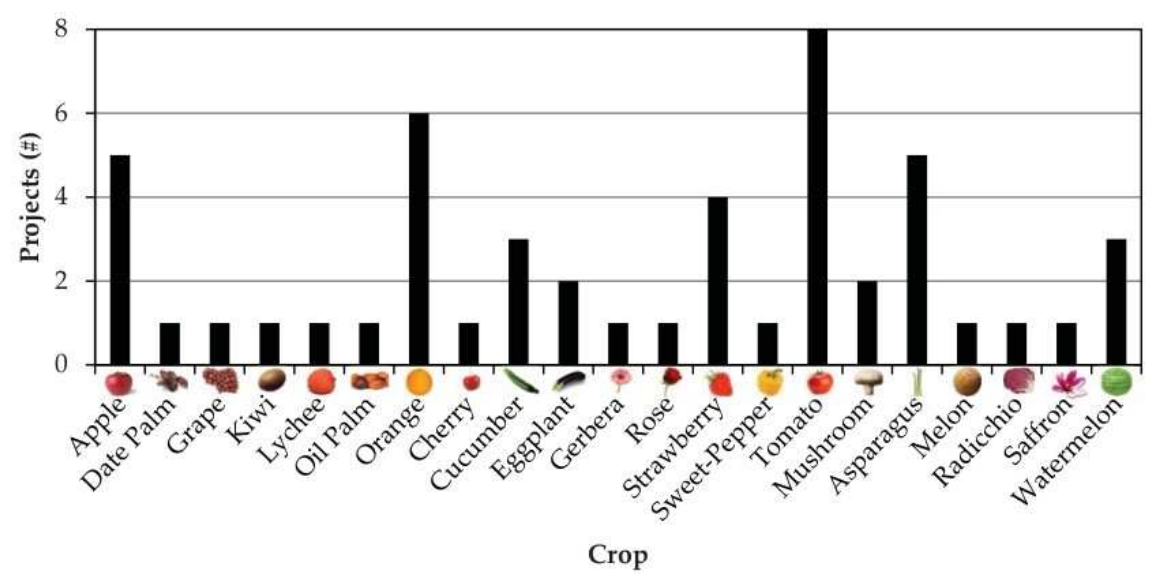


Figura 3: Proyectos existentes sobre robots cosechadores (Bac, van Henten, Hemming, & Edan, 2014)

Estado del arte de visión computacional para la asistencia de robots cosechadores.

La primera referencia ocurre en 1968 con información fotométrica, calculando diferencias entre la reflectividad de las hojas y los frutos en el espectro infrarojo o visible, el autor considera los siguientes problemas:

- iluminación no uniforme
- concentración de follaje

utilizando bandas de reflectancia de 660 nm, se utilizó una cámara B/W con filtro para incrementar el constaste entre rojos y verdes, se calcula la máscara con pixeles brillantes, se filtra con operaciones morfológicas de ruido y se calculan las dimensiones (E. A. Parrish, Jr., & A. K. Goksel, 1977).

(Grand, G., 1987) desarrolla un filtro de manzanas para colores rojos, invariantes de la sombra generada por el árbol, se mejora más tarde utilizando información de la reflectividad.

(Dale Whittaker, G. E. Miles, O. R. Mitchell, & L. D. Gaultney, 1987) desarrolla un sistema para procesar gradientes de intensidad de una imagen en

escalas de grises aplicando transformadas circulares para detectar Figuras circulares, tenia errores con las hojas y la localización no era exacta.

Una compañía italiana AID, desarrolla un prototipo utilizando luz artificial, se calculaba una imagen pseudo gris con información de color naranja, utilizando imágenes con vectores de dirección para la clasificación por forma, el 70% de las frutas era reconocidas (Levi, P., Falla, A., & Pappalardo, R. ,1988).

(David C. Slaughter & Roy C. Harrell, 1987) desarrollan un sistema, utilizando un sistema de luz artificial, segmentado con los canales H y S usando un clasificador linear con un area rectangular y operaciones de banda entre un mínimo y un máximo, más tarde lo ampliaron con valores en RGB y filtros bayesianos.

(R. C. Harrell, P. D. Adsit, T. A. Pool, & R. Hoffman, 1990) menciona los diferentes factores para el que un robot cosechador no pueda ser comercializado con éxito, principalmente por la ineficiencia en la visibilidad, desarrolla más tarde un sistema para estimar el radio y la posición de cada fruta en una imagen, calculando segmentos y más tarde el cómputo de diámetros horizontales y verticales.

(Peter W. Sites & Michael J. Delwiche, 1988) desarrolla un método para reconocer manzanas maduras y duraznos, utilizando un método con filtros de colores para incrementar el contraste entre la fruta y el fondo, se realiza en 5 pasos:

1. umbralado basado en histogramas con un 37% de pixeles válidos.
2. Mejora del segmentado, usando filtros morfológicos
3. etiquetado con el criterio de componentes vecinales 8
4. extracción de características (área, perímetro, compactamiento, elongación y momentos invariables)
5. clasificación usando una función linear o un método de vecino más cercanos

90% de los frutos visibles fueron detectados durante la noche.

En Francia, CEMAGREF investiga el desarrollo de robots cosechadores. se utilizó un filtro de color rojo con una cámara B/W y dos luces estroboscópicas para hacer el ambiente homogéneo, usando un

algoritmo de umbralado rápido, fueron detectados 80% de frutas, pero con alto número de falsos positivos.
Se mejoró utilizando dos cámaras, otra con un filtro verde, eliminando los falsos positivos, en el tercer experimento se utilizó métodos bayesianos con patrones RGB y un éxito del 90% con 5% de error (Juste, F., & Sevila, F., 1992).

(Cardenas-Weber, M., Hetzroni, A., & Miles, G. E., 1991) desarrolla un sistema para localizar en un plano horizontal, computar las coordenadas X,Y del centro de un melón y estimar el tamaño del melón.
Se realizó segmentación por umbralado, extracción de componentes y generación de hipótesis, la textura y la forma fueron computadas para obtener los candidatos finales.

En la segunda etapa se agregó una evaluación usando reglas de conocimiento directo para eliminar ruidos y múltiples ocurrencias, si el siguiente paso no fue empleado, 89% de detección eran detectadas con múltiples falsos positivos, con las reglas basadas en conocimiento se obtuvieron 84% de éxitos y 10% de falsos.

El robot AUFO (Kassay,1992) utilizaba un sistema de visión estéreo que generaba posición tridimensional de cada fruto detectado, este sistema de visión utilizaba dos cámaras separadas y orientadas para converger en la misma escena del árbol.

Una vez tomadas las imágenes se segmentaba, obteniendo superficies pertenecientes a manzanas, las regiones eran agrupadas y se obtenían sus centros geométricos.

Dada la posición X Y del plano horizontal combinando ambos ejes se computaban las posiciones en Z para cada cámara, si la diferencia entre estos eran de más de 40 mm, entonces el objeto era considerado como presente, aunque solo 41% de frutas fueron detectadas con falsos positivos.
Existían dos problemas, se necesitaba revisar los pares, existiendo situaciones donde virtualmente no existieran, además de los problemas de oclusión.

(Dobrusin, Y. et al, 1992) presenta un sistema para la detección de melones, en dos fases, una fase lejana y otra próxima, utilizando un

sistema con una cámara B/W para la detección en coordenadas X, Y de melones junto con la visión cercana con un plano de luz láser para detectar la distancia en Z.
Ocurrían errores debido al viento, las imágenes eran detectadas, segmentadas con umbral, erosión morfológica y operadores OR, se extraían características como forma, área, tamaño con un clasificador basado en reglas, se propuso el uso de imágenes infrarojas para detectar la diferencia entre hojas, frutas y tierra.

(Israel, Benady y Miles, 1992) reportaron un sistema de visión cercana para detectar la posición en Z, utilizando un plano de luz para iluminar la escena, y calculando curvaturas en melones con transformada circular de Hough para extraer el centro en base a características de tamaño, forma, distancia esperada al suelo y altura.

(Plá, Juste, & Ferri, 1993) soluciona el problema de madurez del cítrico cuando la fruta estaba verde, resolvió el problema con superficies convexas, usando luz de unas lámparas y una cámara B/W se definían la intensidad de la imagen y superficies convexas definiendo en gradientes donde pudiera estar la fruta, de esta manera detecta la intensidad de objetos esféricos.

Se calcula el laplaciano de funciones gaussianas (LOG) con la intensidad de la imagen, al resultado se calcula un umbralado de pixeles con cierta curvatura, para los siguientes pasos, se calcula la diferencia entre formas elipsoidales y las formas del umbralado, obteniendo un error. Con un 75% de éxito y un 8% de falsos positivos, debido a manchas de nubes.

(Buemi, 1995), desarrolla un sistema basado en color para jitomates, con información tridimensional de visión estereo y espacio HSI , con éxito del 90%.

(Pla, 1996) propone un sistema de dos pasos: segmentación de contornos con curvaturas y segmentación agrupada con ciertos parámetros (radio, centro y proporción de de contornos visibles), aunque tenia muchos falsos positivos con algunas hojas.

(Jiménez, Jain, Ceres, & Pons, 1999) Propusieron el uso de un sensor láser de distancia, utilizando reflectancia para analizar objetos generalmente esféricos, utilizando diferentes primitivas, como contornos, cabezas,

convexión y reflectancia usado para estimar parámetros tridimensionales de esferas, posición 3D, radio y reflectividad, con 80% de frutas caracterizadas, el sistema era útil ante sombras, cambios de iluminación y la presencia de fondos en el árbol.

Parámetros de desempeño de visión para robots cosechadores.

Se seleccionan ciertos indicadores de desempeño basados en la literatura existente, es importante tener en cuenta estos parámetros de desempeño al momento de la implementación para permitir en un futuro la mejora del sistema y conocer el enfoque de las debilidades del robot cosechador.

- Éxito de localización.
- Porcentaje de falsos positivos.
- Éxito de desprendimiento.
- Éxito de cosecha.
- Tiempo de ciclo.
- Porcentaje de daño.
- Número de frutas evaluadas en la prueba.
- Proporción de intentos de desprendimiento.

El éxito de localización se obtiene con el número de frutas maduras localizadas por número total en la planta, es importante porque se requiere que una fruta sea localizada para ser desprendida.

El porcentaje de falsos positivos es el número de objetos detectados falsos como frutas entre el total de frutas maduras en la planta, es necesario porque puede influir en el número de intentos de desprendimiento fallidos, daño a la fruta e incrementos en los ciclos de trabajo (Figura 4).

El éxito de desprendimiento es el número de frutas cosechadas con éxito entre el total de frutas maduras localizadas, este indicador mide el desempeño de los movimientos requeridos por el robot.

El éxito de cosecha es el número de frutas maduras cosechadas exitosamente entre el numero total de frutas en el arbusto, este indicador mide el éxito general del ciclo de cosecha.

El tiempo de ciclo es el tiempo promedio requerido para realizar una operación completa de cosecha, incluyendo determinación de madurez, localización, desprendimiento de la fruta, transporte y movimiento del robot a la siguiente fruta.

El porcentaje de daño es el número de frutas o pedúnculos dañados entre el total de frutas maduras localizadas, el daño a los pedúnculos se refleja en una disminución en la viabilidad económica del robot.

El número de frutas evaluadas en la prueba es el número de productos evaluados para calcular el éxito de localización, detección de falsos positivos y tiempo de ciclo, su significancia radica en análisis estadísticos.

La proporción de intentos de desprendimiento es el número de intentos de desprendimiento dividido entre el número de desprendimientos exitosos, este parámetro mide el éxito general del proyecto.

De aquí se ha seleccionado un porcentaje promedio de éxito de localización, con un 85% promedio de todos los proyectos de robots cosechadores. (Figura 5, 6, 7)(Bac, van Henten, Hemming, & Edan, 2014).

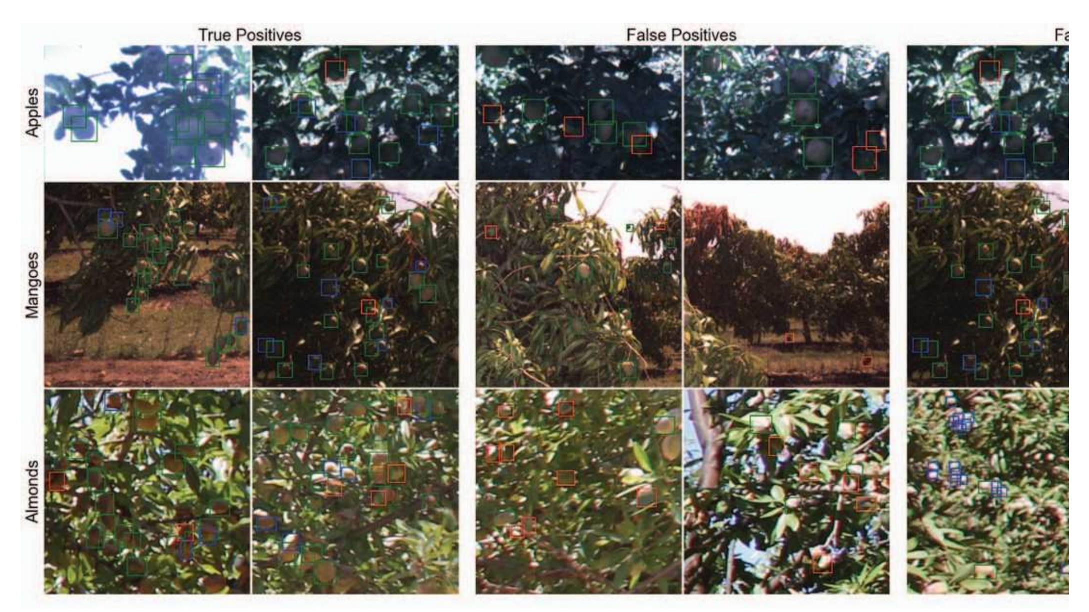


Figura 4: Verdaderos Positivos y Falsos Positivos

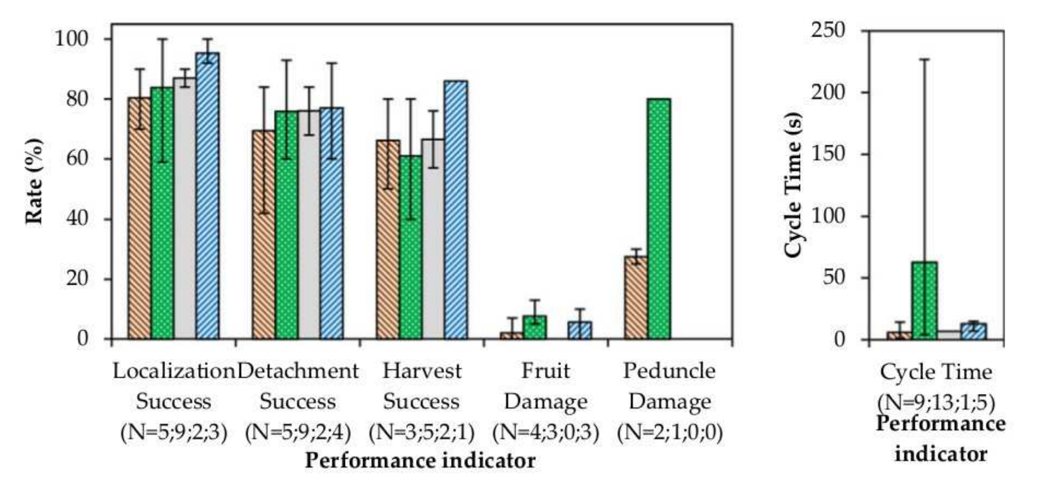


Figura 5: Indicadores agrupados por entornos, huertas, invernaderos, interiores y campo abierto respectivamente (Bac, van Henten, Hemming, & Edan, 2014).

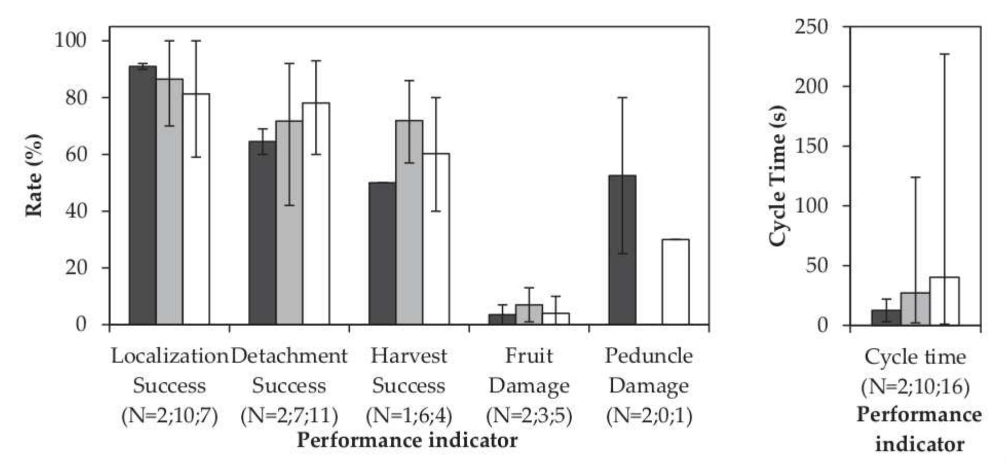


Figura 6: Indicadores agrupados por décadas, 1984-1992, 1993-2002, 2003-2012 respectivamente (Bac, van Henten, Hemming, & Edan, 2014).

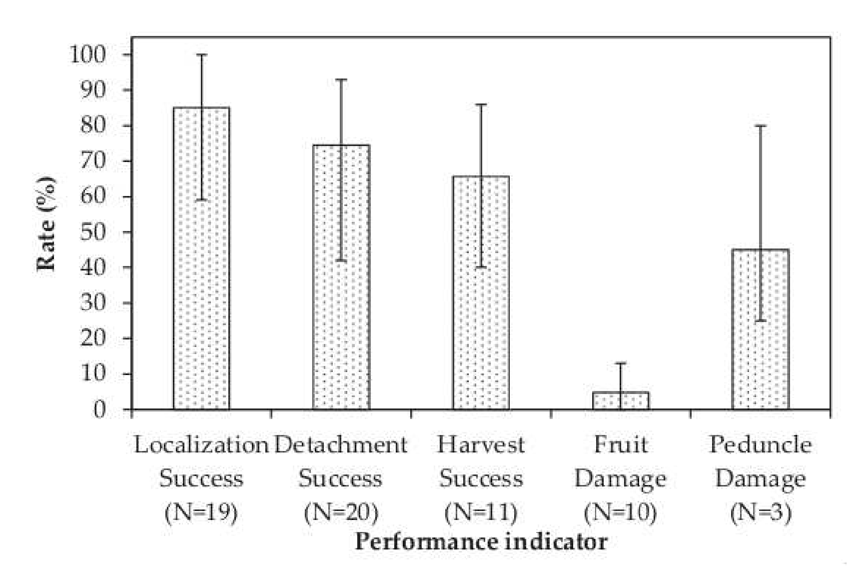


Figura 7: Promedio de indicadores (Bac, van Henten, Hemming, & Edan, 2014).

Técnicas de visión computacional para robots cosechadores.

Múltiples acercamientos se han hecho para la detección y localización de frutas, aunque debido a la variabilidad en la agricultura se ha vuelto muy difícil adaptar algoritmos ya creados.

Se ha propuesto la visión como técnica predominante, debido a que en la práctica, la caracterización del producto se realiza de manera visual, muchas decisiones de creación de algoritmos están basados enteramente en apariencia del producto, utilizando principalmente características como intensidad, color, forma y textura, aunque se han estudiado además, métodos basados en probabilidades e Inteligencia Artificial.

El sistema de visión debe realizar las siguientes tareas:

- Localización de la fruta en el árbol.
- Caracterización de la fruta (madurez) .
- Asistencia de la etapa de desprendimiento.
- Control de navegación en el entorno.

(Bachche, 2015)

El éxito de la visión computacional depende principalmente de que se logren resolver retos relacionados con características del entorno:

- Oclusión generado por hojas, ramas y demás partes de la planta (Figura 10).
- Variación en iluminación y movimiento (Figura 9).
- Variación de forma, tamaño, volumen y disposición de los frutos (Figura 8).
- Sobreposición de frutos.
- Velocidad de procesamiento del sistema.
- Reconstrucción de modelos tridimensionales del mundo

(Bac, van Henten, Hemming, & Edan, 2014)

Por ejemplo se reporta que el humano solo puede observar un 90% de los frutos de un árbol y que el 10% no es posible observarlos debido a diversos factores, como el follaje o la forma en que crecen las plantas. (D. M. Bulanon, T. F. Burks, & V. Alchanatis, 2009), esto es un punto a considerar en la visibilidad que tendrá el robot cosechador y los límites de la visión.

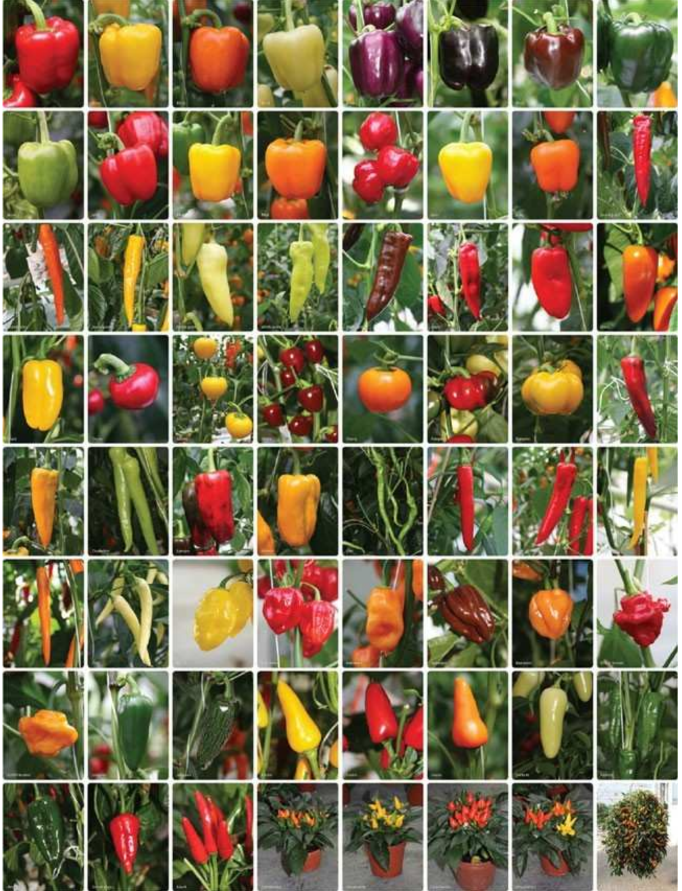


Figura 8: Variabilidad en frutas (Bac, van Henten, Hemming, & Edan, 2014).

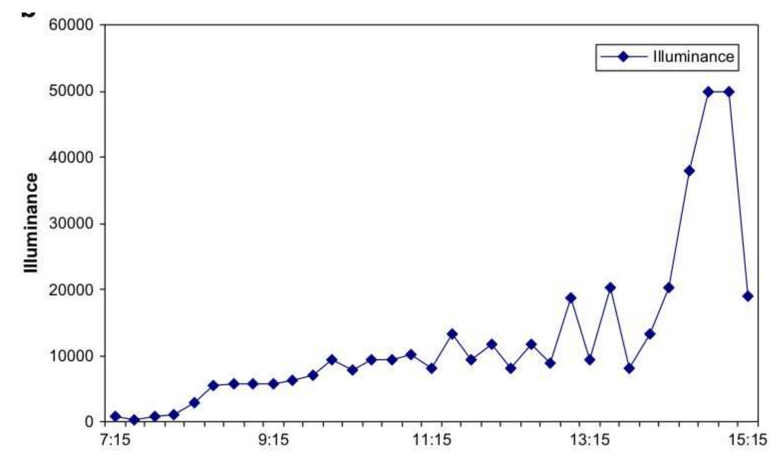


Figura 9: Variabilidad en iluminación (Bulanon, Burks, & Alchanatis, 2009).

Los análisis se pueden clasificar en:

- Análisis locales (color, textura, etc;) (Figura 11).
- Análisis por características geométricas (Figura 12).
- Análisis por Machine learning (Figura 13).

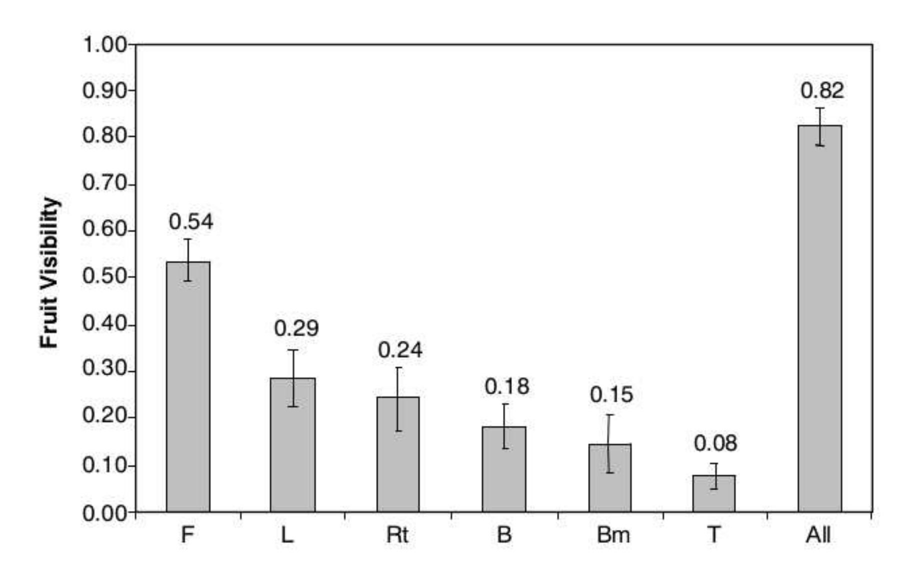


Figura 10: Variabilidad en Visibilidad en frutas (Bulanon, Burks, & Alchanatis, 2009).

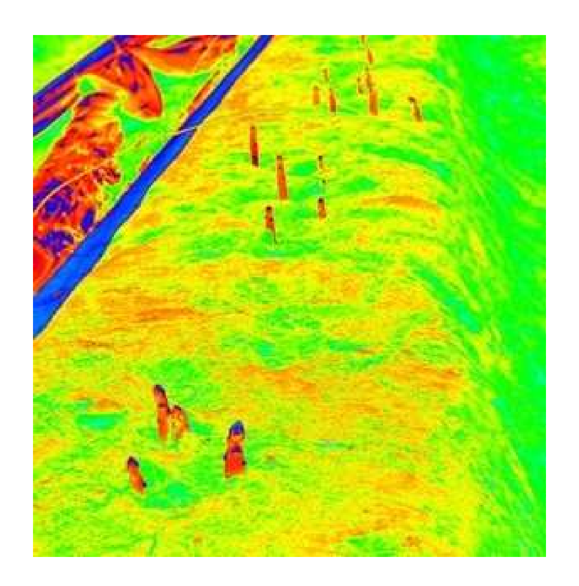


Figura 11: Ejemplo de análisis por color (Chatzimichali, Georgilas, & Tourassis, 2009).

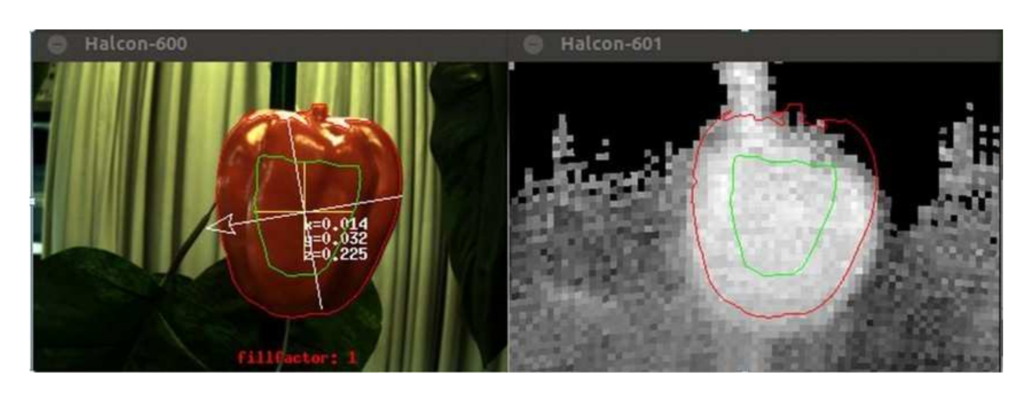


Figura 12: Ejemplo de análisis por geometría (van Henten et al., 2002).

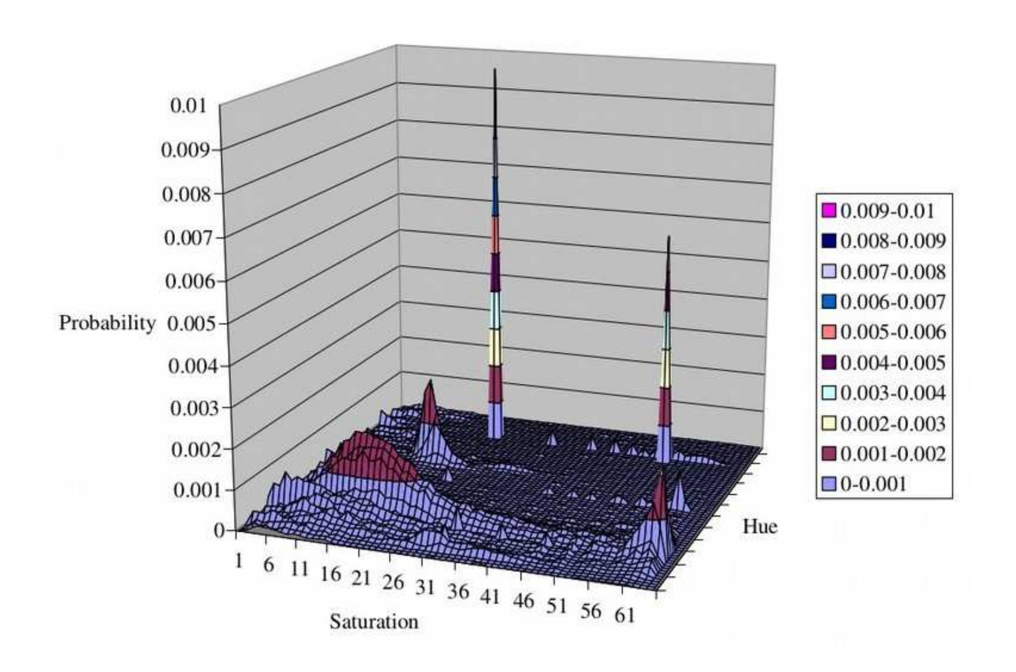


Figura 13: Ejemplo de análisis por machine learning (Murali Regunathan & Won Suk Lee, 2005).

Existen además diversos acercamientos, basados en esquemas, dependiendo del requerimiento de espacio, de exactitud y capacidad de cálculo, se pueden elegir:

- Esquemas monoculares de visión.
- Esquemas binoculares de visión.
- Esquemas de luz estructurada.
- Esquemas multiespectrales de visión.
- Esquemas hiperespectrales de visión.
- Esquemas térmicos de visión.

Los análisis basados en color son los más utilizados, se utilizan principalmente cuando el fruto es distinguible del follaje, suelen utilizar filtros para resaltar características, tanto en espacio de grises como en espacio RGB o HSV (Figura 14), Se han reportado detecciones desde 40% hasta 100% con falsos positivos del 0 al 42%. (D. M. Bulanon, T. F. Burks, & V. Alchanatis, 2009)

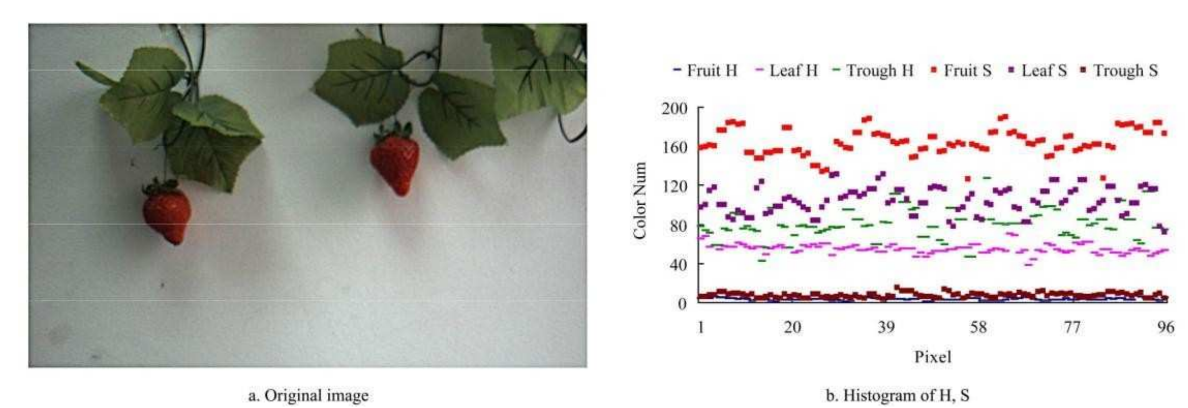


Figura 14: Ejemplo de análisis por color con espacio en HS (Feng, Q., Wang, X., Zheng, W., Qiu, Q., & Jiang, K., 2012).

Los análisis basados en formas suelen utilizarse cuando la forma de la fruta es característica y presenta problemas ante el contraste de color, suelen utilizarse técnicas de transformadas de Hough para calcular centroides, tamaños, orientaciones y formas (Figura 15). Presenta mayor susceptibilidad a interferencias y suele utilizarse además con esquemas de luz estructurada.

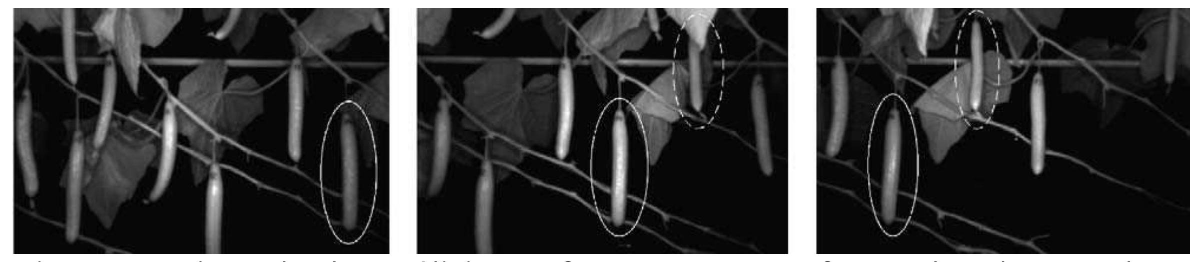


Figura 15: Ejemplo de análisis por forma con transformadas de Hough (Van Henten et al., 2003).

Los análisis basados en estadísticas requieren de un entrenamiento previo, junto con una preparación previa de las características a utilizar, se reportan proyectos con técnicas bayesianas, de clustering, técnicas de discriminantes de Fischer, etc; (Figura 16).

Los proyectos que utilizan esta técnica reportan un 75% de éxito en la detección (Bac, van Henten, Hemming, & Edan, 2014).

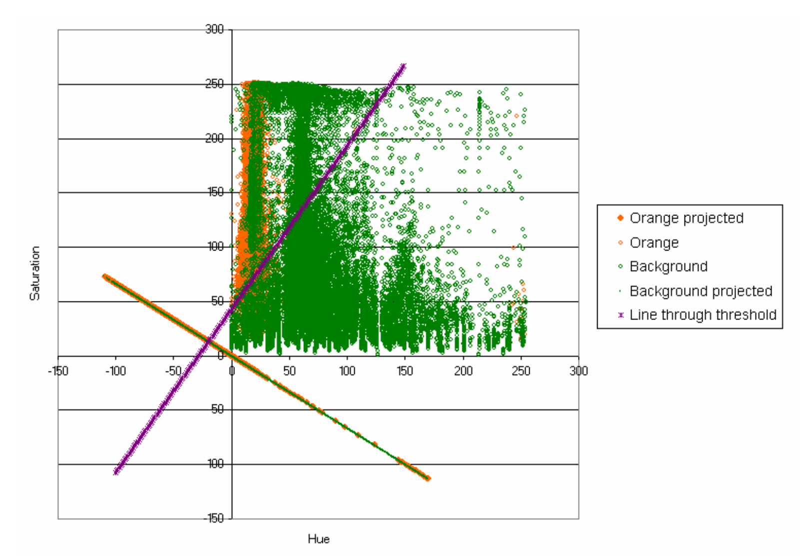


Figura 16: Ejemplo de análisis estadístico con discriminantes de Fischer (Murali Regunathan & Won Suk Lee, 2005).

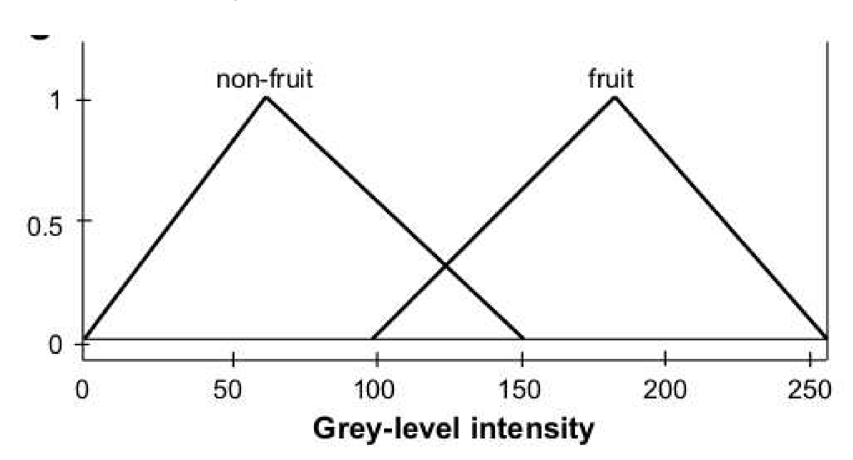


Figura 17: Ejemplo de análisis por Redes neuronales con técnica de lógica difusa en espacio de grises (Bulanon, Burks, & Alchanatis, 2009).

Los análisis por redes neuronales se han utilizado con espacios de colores (tanto HSV como RGB), espacio de texturas, así como técnicas más específicas como análisis por segmentación o por bounding box (Figura 17), estos análisis reportan un 80% de éxito en la detección (Bac, van Henten, Hemming, & Edan, 2014) y se han utilizado además, junto con otras técnicas de inteligencia artificial, como lógica difusa, máquina vector y VSQ.

El esquema monocular de visión funge un papel importante, debido a la facilidad de implementación y al bajo coste por utilizar una sola cámara, aunque puede requerir de sensores externos para los cálculos espaciales (sensores ultrasónicos, lásers, etc;) (Figura 18), se han reportado éxitos variados del 70% al 90% (Bac, van Henten, Hemming, & Edan, 2014).

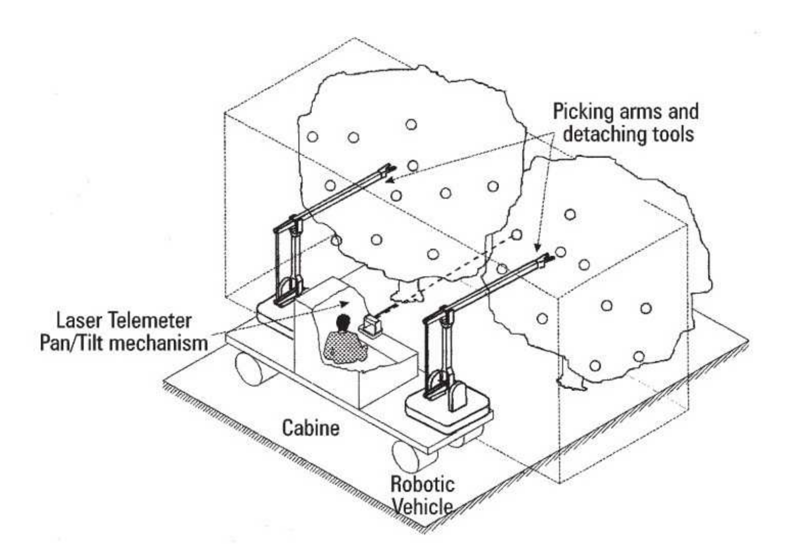


Figura 18: Ejemplo de esquema monocular con sensor de distancia por láser.

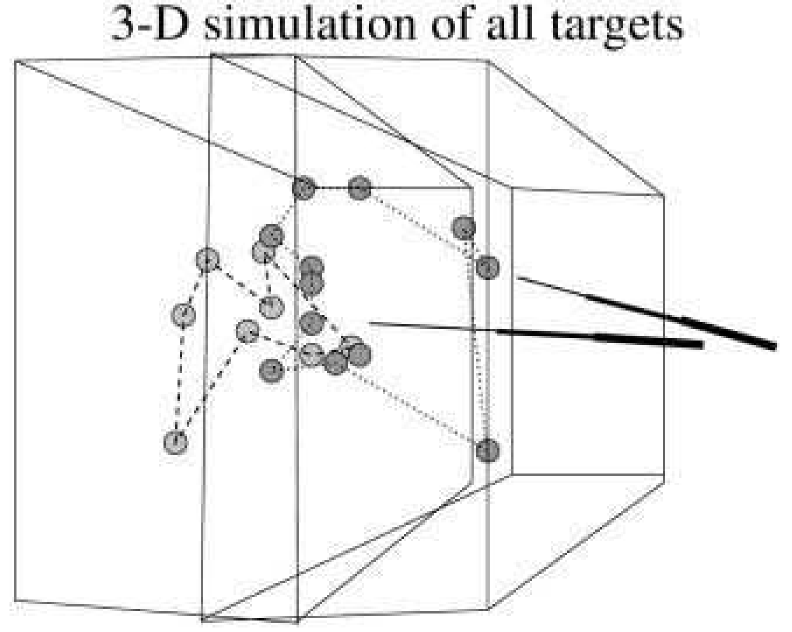


Figura 19: Ejemplo de esquema binocular con modelo tridimensional de las frutas (Plebe & Grasso, 2001).

Los esquemas binoculares de visión, suelen utilizarse cuando se requiere de un cálculo más preciso del espacio, utilizando dos o más cámaras y principios de triangulación o fotogametría con el inconveniente de requerir de mayor tiempo de procesamiento y enfrentarse con problemas de iluminación (Figura 19).

Los esquemas de luz estructuradas utilizan proyecciones de luz para calcular el espacio tridimensional, puede utilizarse en conjunto con dos cámaras, aunque puede presentar errores por orientación y oclusión de hojas (Figura 20), se reporta un 87% de éxito en la detección (Bac, van Henten, Hemming, & Edan, 2014).

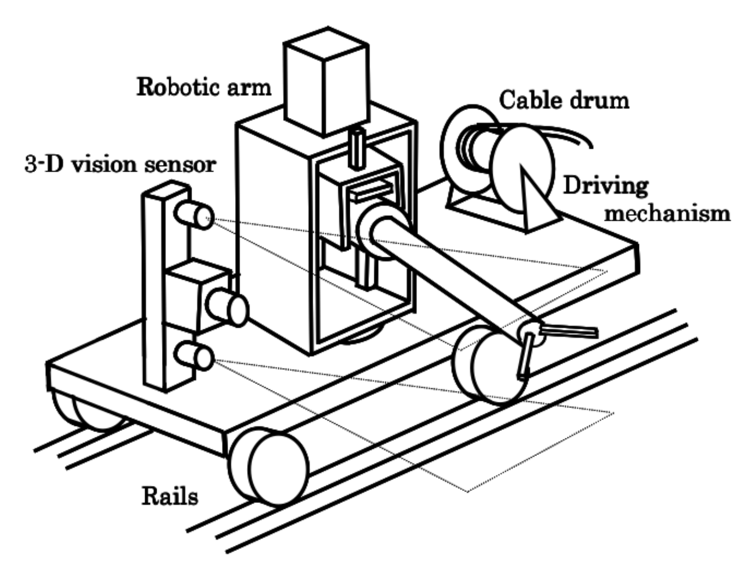


Figura 20: Ejemplo de esquema de luz estructurada con modelo por lasers (Irie, Taguchi, Horie, & Ishimatsu, 2009).

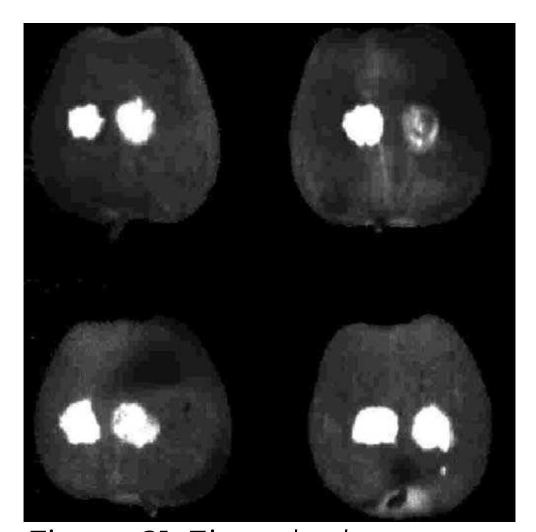


Figura 21: Ejemplo de esquema hiperespectral con calificación de enfermedades en manzanas (Chen, Chao, & Kim, 2002).

Los esquemas hiperespectrales funcionan esparciendo ondas de diversas bandas del espectro, suele utilizarse para la calificación de calidad,

presentan los mismos problemas que los esquemas monoculares (Figura 21).

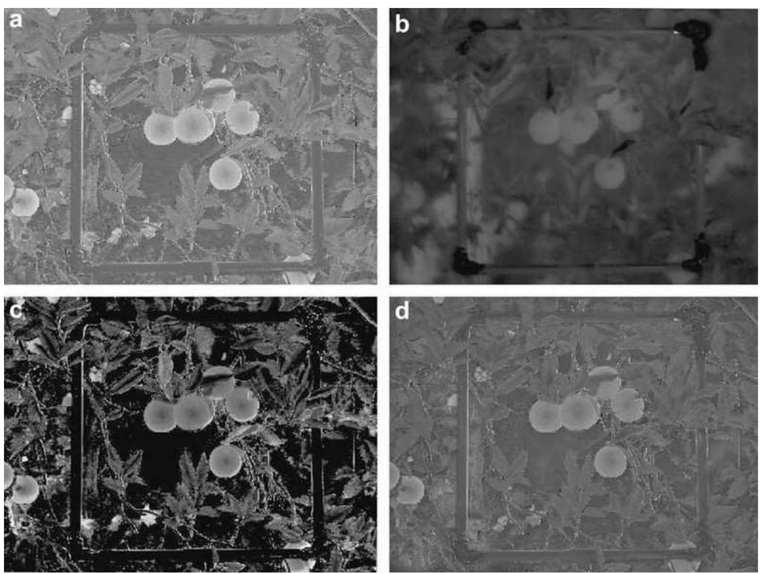


Figura 22: Ejemplo de esquema térmico y problemas con variación de temperatura a lo largo del día (Bulanon, Burks, & Alchanatis, 2009).

Se han utilizado además esquemas térmicos, que presentan inconvenientes debido a la variación de temperatura del ambiente y requieren de ambientes controlados (Figura 22).

Los esquemas multiespectrales son similares a los esquemas hiperespectrales, con la diferencia de que no dispersan ondas, sino que las captan, funcionan con ondas cercanas al espectro visible (Figura 23).

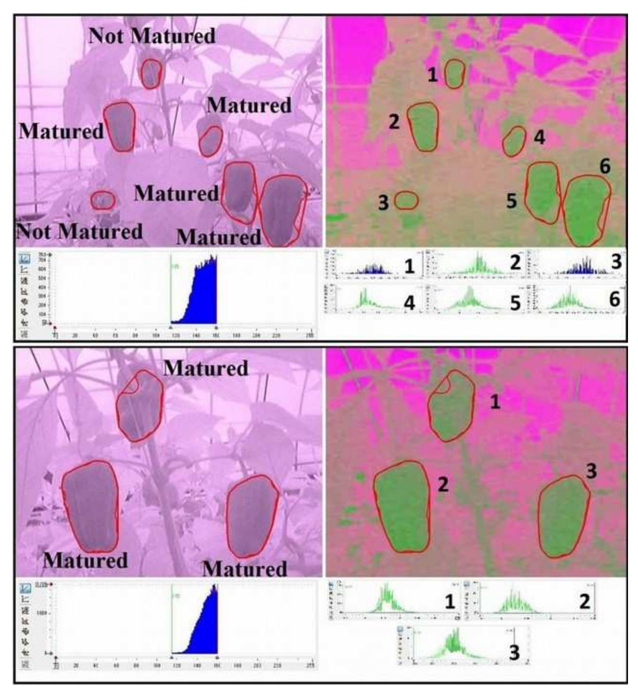


Figura 23: Ejemplo de esquema multiespectral con calificación de madurez (Bachche, 2015).

Técnicas de visión computacional basadas en redes neuronales.

Se han utilizado estas técnicas para diversas tareas como por ejemplo:

- Conteo.
- Clasificación.
- Detección de enfermedades.
- Reconocimiento.
- Detección de madurez.

(Chen et al., 2017)

Se han comparado los resultados de clasificadores neuronales con otros, como los bayesianos y discriminantes como los de Fischer (Figura 24), demostrando mejores resultados en la estimación del tamaño de la fruta y la localización. (Murali Regunathan & Won Suk Lee, 2005)

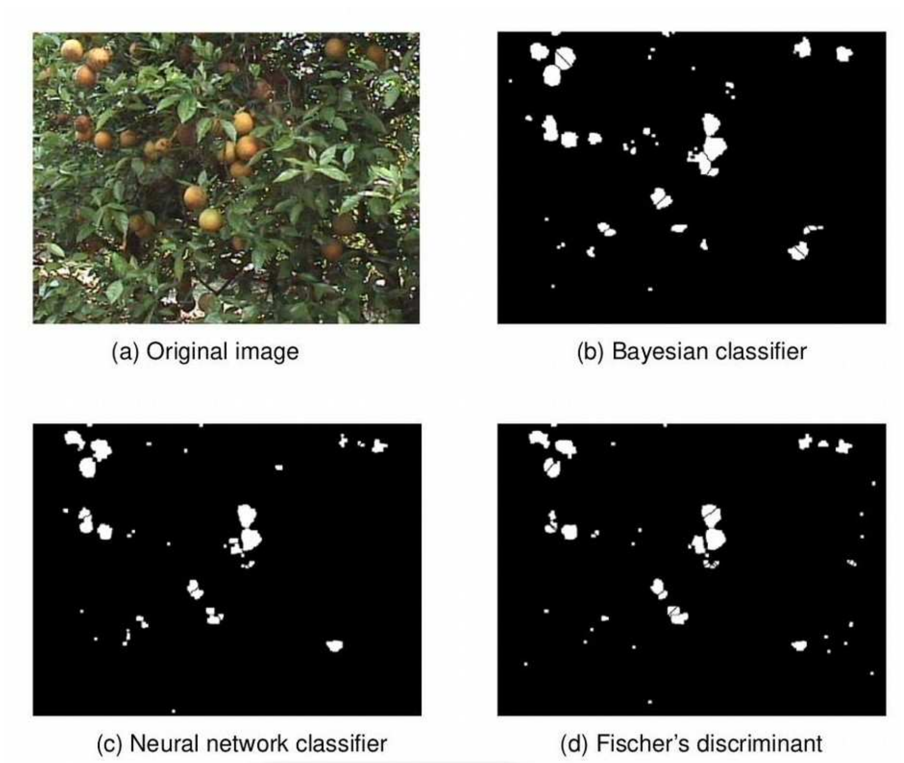


Figura 24: Comparación de clasificadores con redes neuronales (Bulanon, Burks, & Alchanatis, 2009).

Se han presentado múltiples propuestas de perceptrones multicapa, con datos obtenidos del espacio RGB, HSV, mezcla de ambos, datos fusionados de temperaturas, tamaños y otros datos obtenidos por en procesos de umbralados o transformaciones de imagen.

Se han presentado además, propuestas con redes neuronales convolucionales, teniendo mayor preferencias estas, por la fácil implementación y que no requiere de un procesado previo, el éxito de las redes neuronales convolucionales radica en la capacidad de explotar las capacidades de aprendizaje para problemas estructurados de probabilidad.

Existen de acuerdo a la aplicación al menos dos tipos de implementaciones para las redes neuronales Convolucionales (Figura 25):

- A nivel pixel.
- A nivel región.

(Häni, Roy, & Isler, 2018)

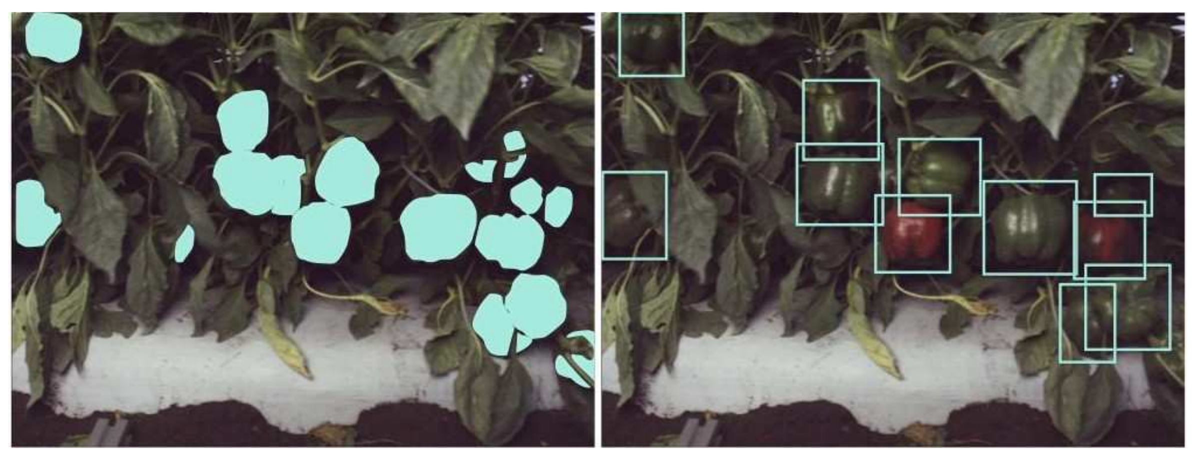


Figura 25: Implementación de redes neuronales a nivel pixel y a nivel región respectivamente (Sa et al., 2016).

La implementación a nivel pixel se utilizan con técnicas de segmentación, que permiten la clasificación punto a punto, utilizando una red completamente conectada, o con segmentación basada en superpíxeles.

Se han venido presentando avances en detección de objetos debido al éxito de redes neuronales de propuesta de Región y de redes convolucionales basadas en regiones, aunque las propuestas de regiones fueron computacionalmente más costosas, se han venido presentando mejoras debido al compartimiento de capas, el algoritmo Fast R-CNN propone éxitos en tiempo cercanos a la realidad.

La implementación a nivel región pueden resolverse de distintas formas, por ejemplo utilizando propuestas de bounding box, propuestas de regiones, etc;

RPN (Region Proposal Network) Resuelve el problema aplicando redes neuronales profundas con la red de objeto de manera interna, entonces el sistema puede predecir límites y clasificarlos en cada posición, los parámetros de las redes son compartidos lo que la hace disponible para fines robóticos (Figura 26).

Con Faster R-CNN se utiliza el formato de color en RGB utilizando propuestas de segmentación con regiones más grandes, usando esquemas por ejemplo de segmentación por superpixeles (Figura 27).

Esta técnica consiste en dos partes, propuesta de regiones y clasificador de regiones.

La propuesta de regiones genera cajas donde los objetos de interés pudieran estar y el clasificador de regiones determina si la región pertenece a un objeto de interés. Para este entrenamiento se requiere un etiquetado por cajas.

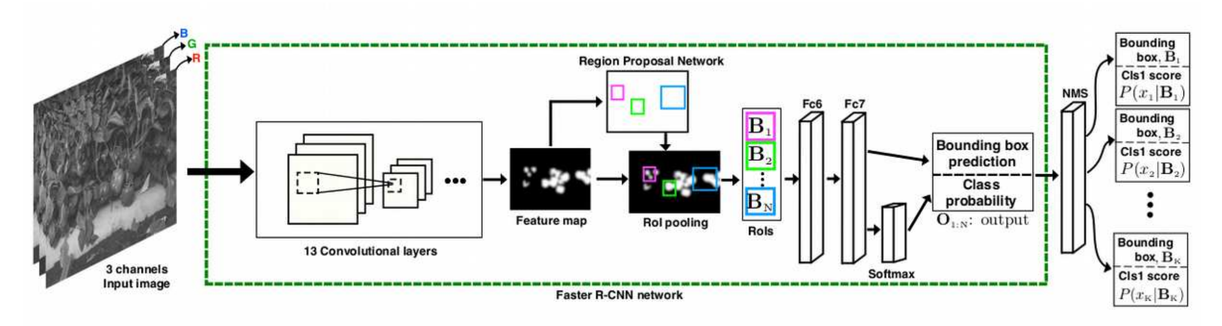


Figura 26: Arquitectura Region Proposal Network (Sa et al., 2016).

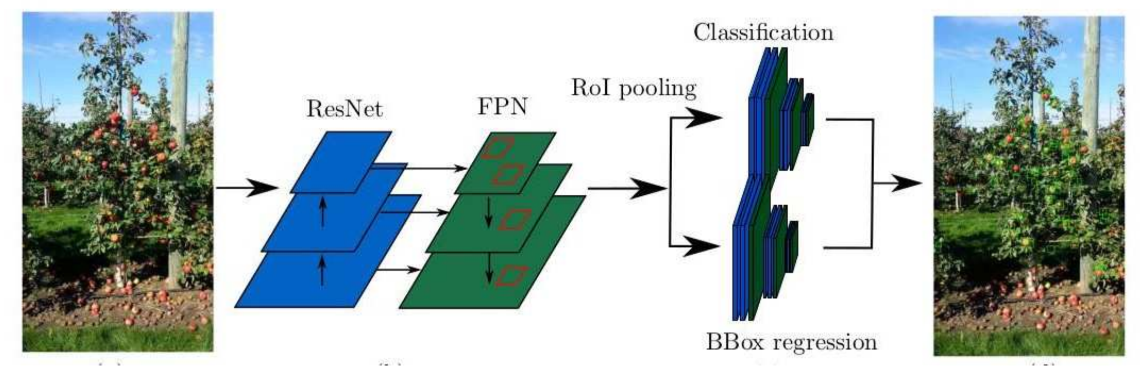


Figura 27: Arquitectura Fast Regional Proposal Network (Häni, Roy, & Isler, 2018).
(Häni, Roy, & Isler, 2018)

Single shot detector: son los detectores que utilizan una sola red neuronal convolucional para predecir clases y anchors sin requerir una operación de propuesta de clasificación, bajo esta definición, Multibox y Region Proposal NN forman parte de este tipo.

Con Region based Fully Convolutional Neural Network en lugar de tomar las características de la misma capa donde las propuestas de regiones se hacen, se toman de la última capa de características antes de la predicción, esto minimiza la cantidad de computo por región que se debe realizar.
(Huang et al., 2016)

Las redes neuronales convolucionales usadas a nivel pixel se compone de dos estructuras principalmente, una parte codificadora y una parte decodificadora (Figura 28).

La parte codificadora se encarga de extraer información de características de una imagen reduciendo espacialmente los datos, sacrificando la cantidad de datos espaciales requeridos, aumentando las capas de mapas de características.

La parte decodificadora se encarga de asignar información espacial a características extrapolando la información, disminuyendo las capas de mapas de características.

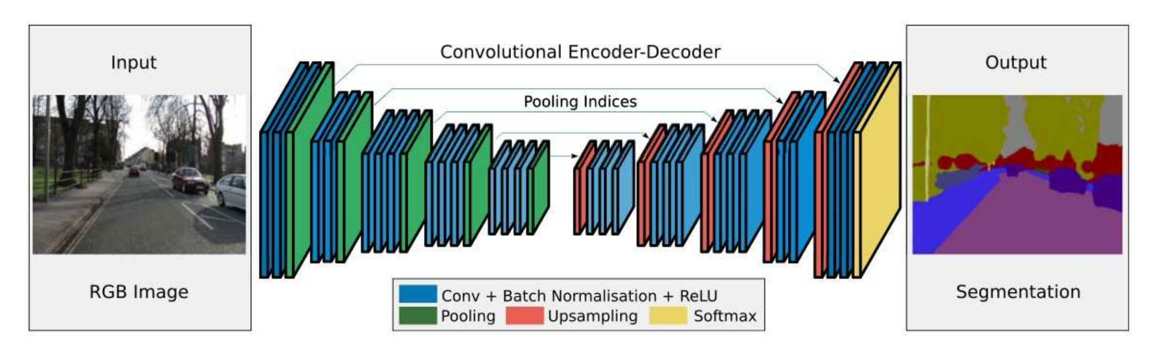


Figura 28: Ejemplo de red neuronal para segmentación (Badrinarayanan, Kendall, & Cipolla, 2017).

Muchos intentos se han realizado por tener segmentación nivel pixel a pixel, esto se ha vuelvo complicado debido a que el max pooling y el subsampleo reducen la resolución de mapas de características.

Para Solucionar esto, muchas arquitecturas se han presentado con arreglos “cortocircuitados” es decir, recuperando información espacial de capas anteriores.
(Huang et al., 2016)

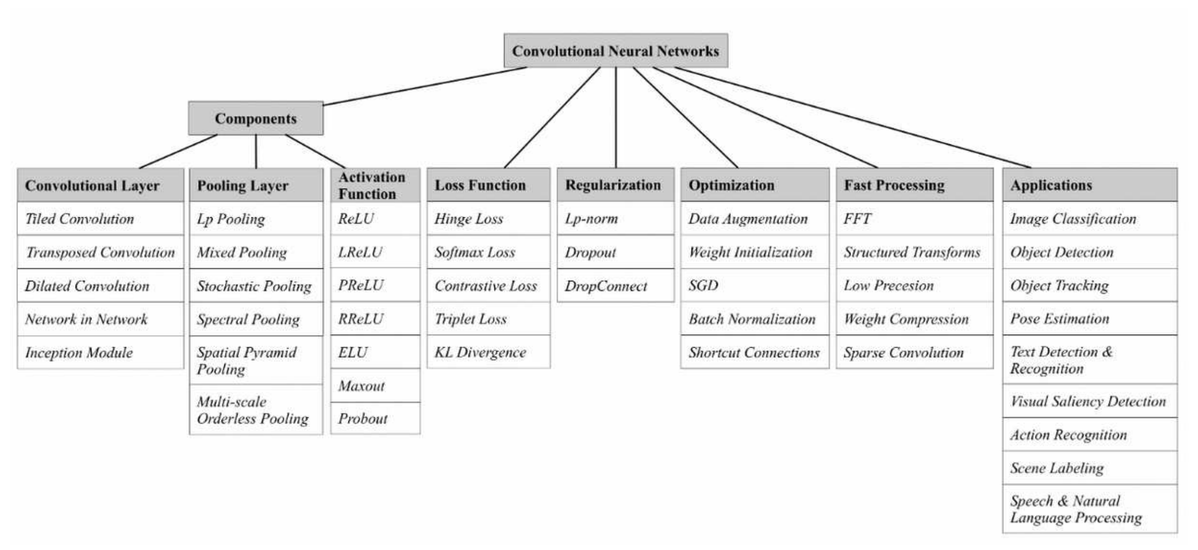


Figura 29: Partes de las redes neuronales convolucionales (Gu et al., 2018).

Las redes convolucionales (Figura 29, 30) están compuestas principalmente de 3 tipo de capas: las capas convolucionales, las capas pooling y las capas completamente conectadas.

Las capas convolucionales son las que aprenden las representaciones de las características de las entradas. Cada nuevo mapa de característica se genera aplicando convoluciones a la última capa y aplicando una función de activación pixel a pixel

La función de activación suele ser generalmente tanh o relu y funciona para agregar no linealidades al sistema.

Las capas tipo pooling para eliminar la variación a los cambios y reducir la resolución de las características, generalmente es una función de máximo o promedio.

Las capas más cercanas a las entradas, detectan características más generales y las más pequeñas características más detalles.

Al final existe una capa completamente conectada que realiza razonamiento a alto nivel (Gu et al., 2018).

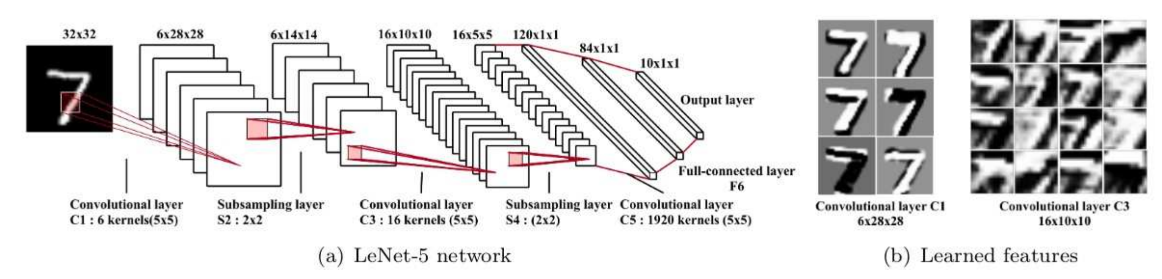


Figura 30: Ejemplo de arquitectura neuronal convolucional (Gu et al., 2018).

Se han presentado casos donde se utilizan arquitecturas con ciertos beneficios:

- Se permite la reutilización de datos de entrenamientos previos (Figura 31).
- Se permite el mejoramiento de la arquitectura y resulta más fácil el entrenamiento.
- Permite tener menores problemas de implementación.

(Zheng, Kong, Jin, Wang, & Zuo, 2019)

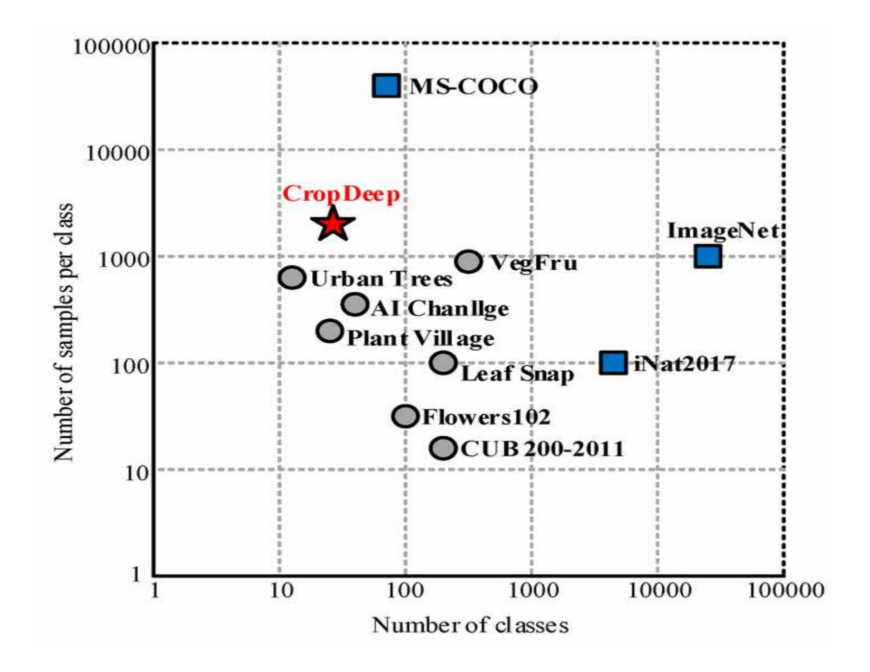


Figura 31: Comparación de algunos datasets existentes (Zheng, Kong, Jin, Wang, & Zuo, 2019).

Para el caso de redes convolucionales usadas para tareas de segmentación se calculan ciertos parámetros que permiten evaluar a la red (Figura 32).

True Positive (TP) se refiere a los verdaderos positivos, blobs de pixeles que se encuentran tanto en la máscara como en la detección.

False Positive (FP) se refiere a los blobs de pixeles que fueron detectados y que no se encuentran en la máscara.

False Negative (FN) se refiere a los blobs de pixeles que no fueron detectados y sí se encuentran en la máscara.

True Negative (TN) se refiere a los blobs de pixeles que no fueron detectados y no se encuentran en la máscara.

Recall = TP / (TP + FN)

Precision = TP / (TP + FP)
F1 = 2 * (Precision * Recall) / (Precision + Recall)

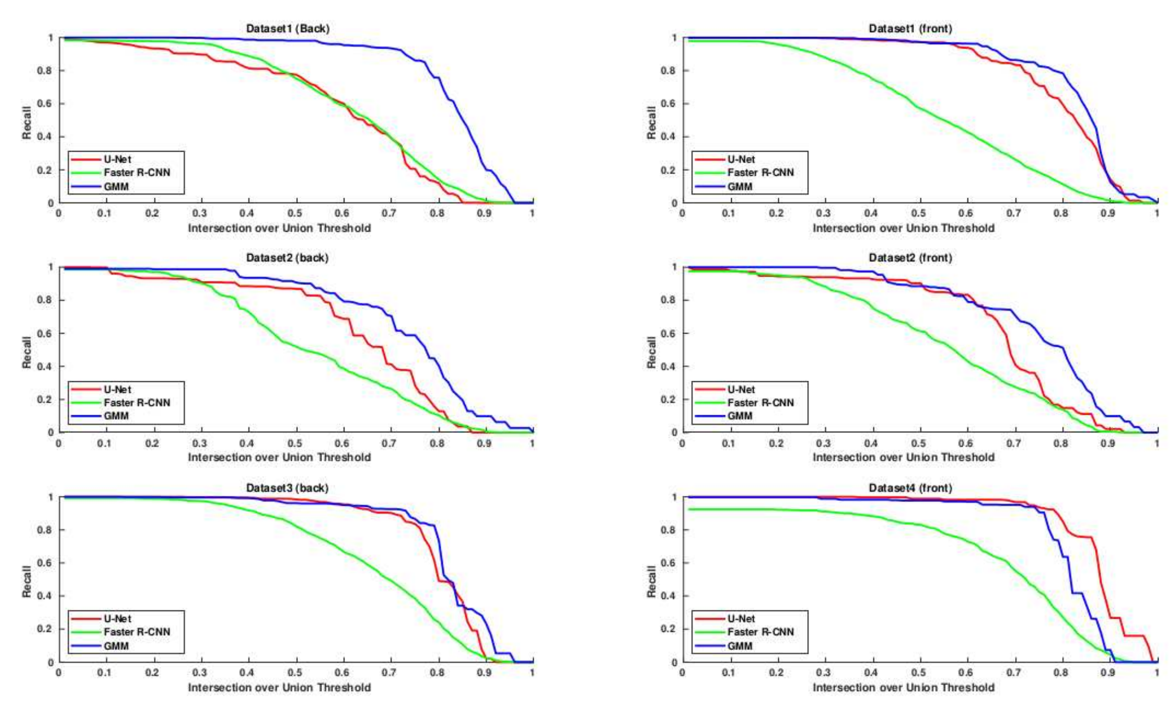


Figura 32: Algunos gráficos de desempeño (Häni, Roy, & Isler, 2018).

En un estudio de (Häni, Roy, & Isler, 2018) se comparan 4 distintas técnicas (U-Net, Faster R-CNN, GMM semisupervisado, GMM supervisado) con mejores resultados con técnicas GMM seguido de técnicas de U-Net.

Propuesta de implementación

Se propone:

- Resolución de los problema de visión de detección, localización y calificación de jitomates maduros.
- La utilización de una red neuronal convolucional, debido a los parámetros de desempeño arriba del promedio.
- Análisis a nivel pixel, realizando segmentación semántica.

- Se eligió utilizar U-Net por la facilidad de aprendizaje y la posible reutilización de datos, debido a que puede trabajar con una cantidad mínima de datos (Ronneberger, Fischer, & Brox, 2015) Se eligió entre otras arquitecturas diseñadas para aprendizaje profundo como ResNet y VGG debido a su diseño sencillo.
- La creación de un dataset de segmentación de jitomates para el aprendizaje de una red neuronal convolucional, en caso de que no existiera un dataset con estas características.
- Aumento de los datos para el aumento de la invariabilidad en la clasificación, bajo distintas condiciones de iluminación y movimientos.
- Modificación de la arquitectura para presentación de nuevos resultados y evaluación de los mismos.

Modelo original

El modelo original utilizado es una arquitectura U-Net, pensada para la segmentación de imágenes biomédicas, el modelo consta de dos partes importantes, una parte codificadora, encargada de generar mapas de características y una parte decodificadora, encargada de asignar las características a una clase a nivel pixel y salida binaria.

Internamente la parte codificadora se compone de un conjunto de capas convolucionales sin padding con kernel 3x3 con maxpooling que se encarga de ir aumentando el número de canales de características mientras disminuye el tamaño de la imagen.

Al final de la extracción de características, se tiene en la parte decodificadora, capas deconvolucionales donde se concatena información de las características de capas anteriores recortadas por el centro para tener el tamaño indicado véase (Figura 33).

Debido a la falta de padding en la red y a los recortes centrados de las capas, la red genera una salida que es equivalente a un ROI en el centro de la imagen original (Figura 34).

Además, la red neuronal tiene un preprocesado de las máscaras, añadiendole la imágen original con los bordes resaltados, para forzar ala red a aprender separación entre pixeles (Figura 35).

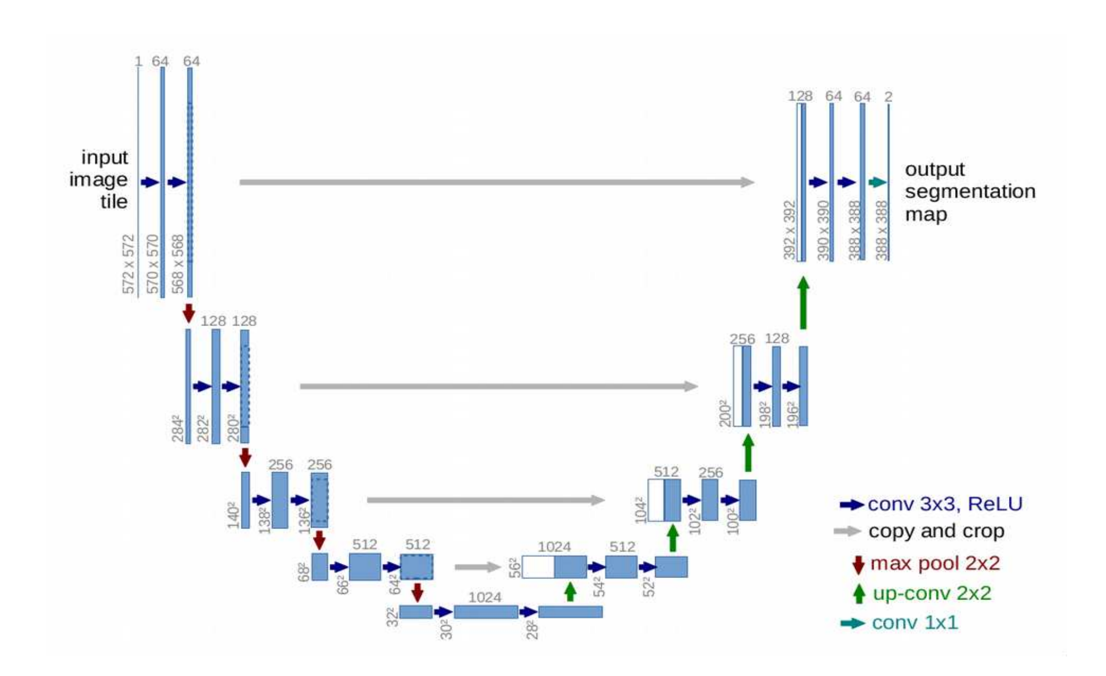


Figura 33: Arquitectura U-Net original (Ronneberger, Fischer, & Brox, 2015).

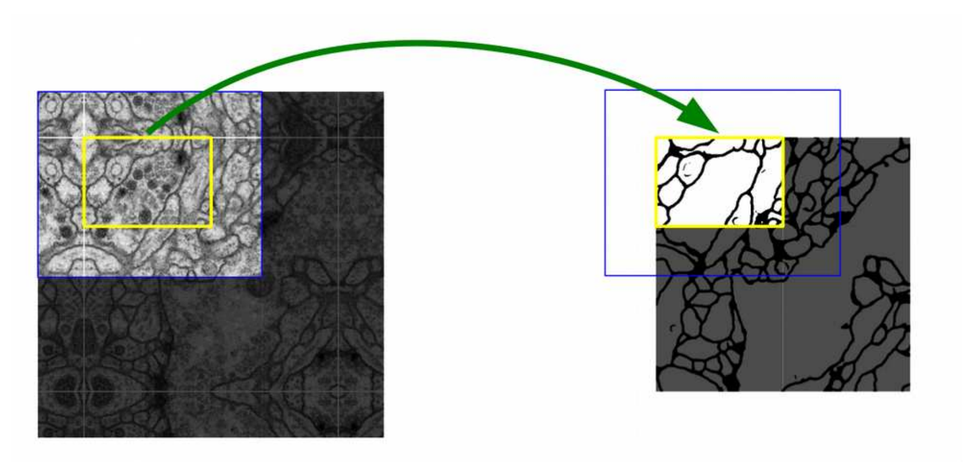


Figura 34: Centrado en la salida de la red U-Net (Ronneberger, Fischer, & Brox, 2015)

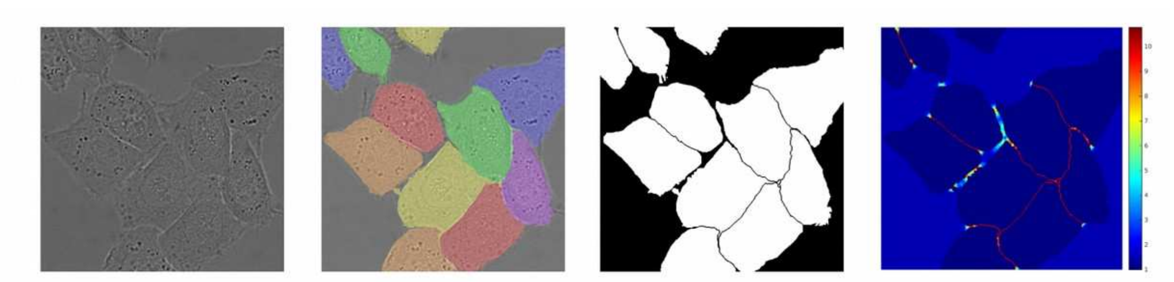


Figura 35: borde añadido a las máscaras de etiquetado (Ronneberger, Fischer, & Brox, 2015).

Modelo propuesto

la forma propuesta para la arquitectura se describe en la tabla 1 y se muestra de manera sintética en la Figura 36.

no.	operaciones	Dimensiones resultantes (B, N, R, G ,B)
1	Input	B, 3, 572, 572
2	Conv2d 3 capas de entrada, 64 capas de salida, kernel (3x3), sin padding,	B 64 570 570
3	Normalización de lote	
4	LeakyRelu	
5	Conv2d 64 capas de entrada, 128 capas de salida, kernel (3x3), sin padding,	B 128 568 568
6	Normalización de lote	
7	LeakyRelu	
8	Maxpooling kernel (2,2), stride (2,2)	B 128 284 284
9	Conv2d 128 capas de entrada, 128 capas de salida, kernel (3x3), sin padding,	B 128 282 282

10	Normalización de lote	
11	LeakyRelu	
12	Conv2d 128 capas de entrada, 256 capas de salida, kernel (3x3), sin padding,	B 256 280 280
13	Normalización de lote	
14	LeakyRelu	
15	Maxpooling kernel (2,2), stride (2,2)	B 256 140 140
16	Conv2d 256 capas de entrada, 256 capas de salida, kernel (3x3), sin padding,	B 256 138 138
17	Normalización de lote	
18	LeakyRelu	
19	Conv2d 256 capas de entrada, 512 capas de salida, kernel (3x3), sin padding	B 512 136 136
20	Normalización de lote	
21	LeakyRelu	
22	Maxpooling kernel (2,2), stride (2,2)	B 512 68 68
23	Conv2d 256 capas de entrada, 512 capas de salida, kernel (3x3), sin padding,	B 512 66 66
24	Normalización de lote	
25	LeakyRelu	
26	Conv2d 512 capas de entrada, 1024 capas de salida, kernel (3x3), sin padding,	B 1024 64 64
27	Normalización de lote	
28	LeakyRelu	
29	Maxpooling kernel (2,2), stride (2,2)	B 1024 32 32
30	Conv2d 1024 capas de entrada, 1024 capas de salida, kernel (3x3), sin padding,	B 1024 30 30
31	Normalización de lote	
32	LeakyRelu	

33	Conv2d 1024 capas de entrada, 2048 capas de salida, kernel (3x3), sin padding,	B 2048 28 28
34	Normalización de lote	
35	Tanh	
36	Interpolación de tamaños W,H de la capa anterior al doble, modo bilinear	B 2048 56 56
37	Obtención de los primeros 1024 canales de la capa anterior	B 1024 56 56
38	Recorte centrado de la capa que mide B 1024 64 64 de un ROI de 56 x56	B 1024 56 56
39	Concatenación de los canales de las dos capas anteriores [B 1024 56 56 , B 1024 56 56]	(B 1024 56 56 + B 1024 56 56) B 2048 56 56
40	Normalización de lote	
41	LeakyRelu	
42	Conv2d 2048 capas de entrada, 1024 capas de salida, kernel (3x3), sin padding,	B 1024 54 54
43	Normalización de lote	
44	LeakyRelu	
45	Conv2d 1024 capas de entrada, 1024 capas de salida, kernel (3x3), sin padding,	B 1024 52 52
46	Normalización de lote	
47	LeakyRelu	
48	Interpolación de tamaños W,H de la capa anterior al doble, modo bilinear	B 1024 104 104
49	Obtención de los primeros 512 canales de la capa anterior	B 512 104 104
50	Recorte centrado de la capa que mide B 512 136 136 de un ROI de 104 x 104	B 512 104 104

51	Concatenación de los canales de las dos capas anteriore [B 512 104 104 , B 512 104 104]	(B 512 104 104 + B 512 104 104) B 1024 104 104
52	Normalización de lote	
53	LeakyRelu	
54	Conv2d 1024 capas de entrada, 512 capas de salida, kernel (3x3), sin padding,	B 512 102 102
55	Normalización de lote	
56	LeakyRelu	
57	Conv2d 512 capas de entrada, 512 capas de salida, kernel (3x3), sin padding,	B 512 100 100
58	Normalización de lote	
59	LeakyRelu	
60	Interpolación de tamaños W,H de la capa anterior al doble, modo bilinear	B 512 200 200
61	Obtención de los primeros 256 canales de la capa anterior	B 256 200 200
62	Recorte centrado de la capa que mide B 256 280 280 de un ROI de 200 x 200	B 256 200 200
63	Concatenación de los canales de las dos capas anteriores [B 256 280 280 , B 256 280 280]	(B 256 200 200 + B 256 200 200) B 512 200 200
64	Normalización de lote	
65	LeakyRelu	
66	Conv2d 512 capas de entrada, 256 capas de salida, kernel (3x3), sin padding,	B 256 198 198
67	Normalización de lote	
68	LeakyRelu	
69	Conv2d 256 capas de entrada, 256 capas de salida, kernel (3x3), sin padding,	B 256 196 196

70	Normalización de lote	

71	LeakyRelu	
72	Interpolación de tamaños W,H de la capa anterior al doble, modo bilinear	B 256 392 392
73	Obtención de los primeros 128 canales de la capa anterior	B 128 392 392
74	Recorte centrado de la capa que mide B 128 568 568 de un ROI de 392 x 392	B 128 392 392
75	Concatenación de los canales de las dos capa anteriores [B 128 392 392 , B 128 392 392]	(B 128 392 392 + B 128 392 392) B 256 392 392
76	Normalización de lote	
77	LeakyRelu	
78	Conv2d 256 capas de entrada, 128 capas de salida, kernel (3x3), sin padding,	B 128 390 390
79	Normalización de lote	
80	LeakyRelu	
81	Conv2d 128 capas de entrada, 128 capas de salida, kernel (3x3), sin padding,	B 128 388 388128
82	Normalización de lote	
83	LeakyRelu	
84	Conv1d 128 capas de entrada, 3 capas de salida, kernel (1x1), sin padding,	B 3 388 388
85	Softmax	B 3 388 388

Tabla 1: Capas de la red propuesta

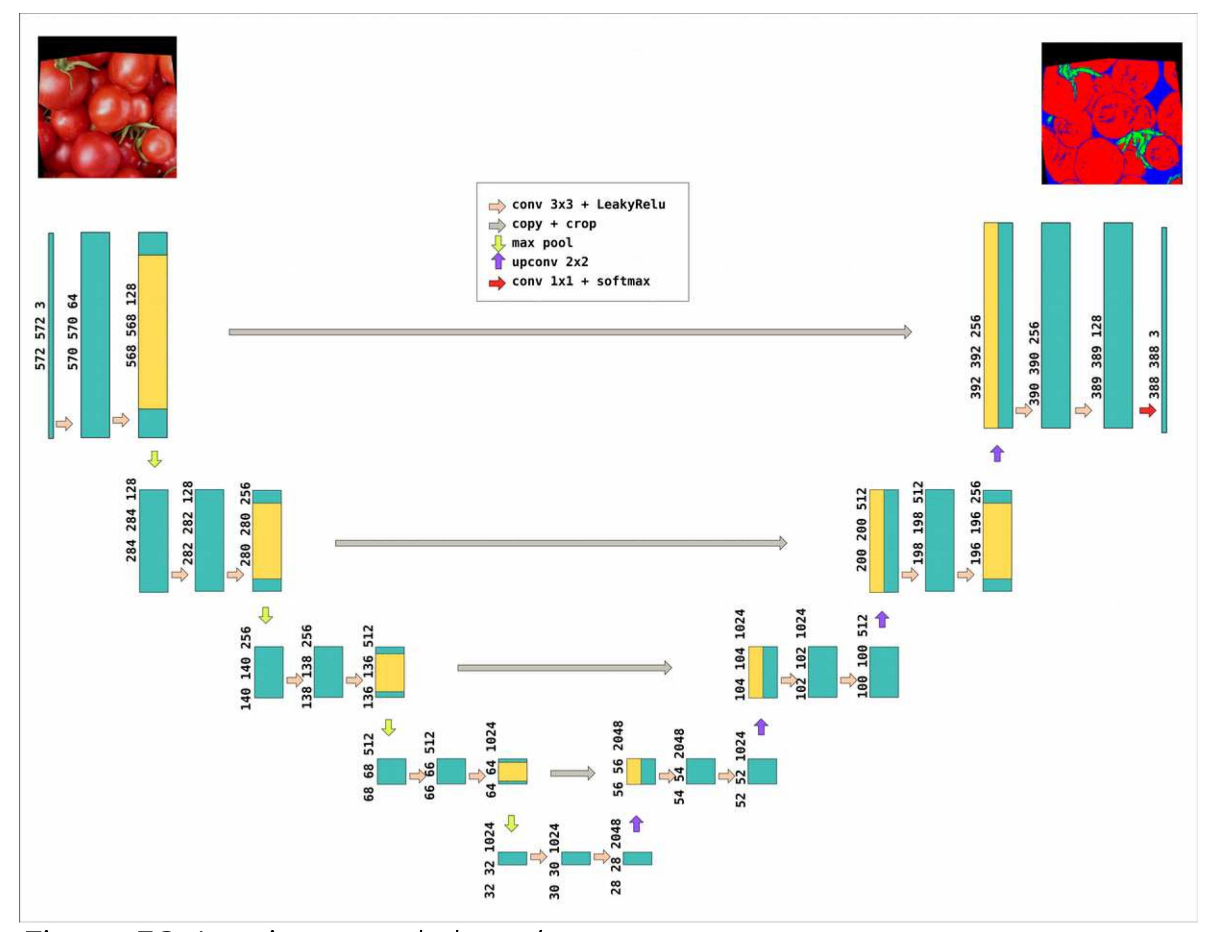


Figura 36: Arquitectura de la red propuesta

Modificaciones a la arquitectura:

Se modificaron las capas de entrada y de salida para coicidir con el número de canales RGB (de densidad 1 a densidad 3).
Se aumentó al doble el número de mapas de características.

Se cambió la función de activación por una función de activación LeakyRelu, que de acuerdo con (Gu et al., 2018) mejora las activaciones de las capas, haciendo el entrenamiento más rápido.
Se agregó normalización de lote que permite aumentar la proporción de aprendizaje, acelerando el proceso de entrenamiento (Ioffe & Szegedy, 2015).

Operación de pérdida:

se utilizó la función de pérdida basada en el Coeficiente de Sorensen-Dice, se cambió la función de pérdida original (Binary Cross Entropy) debido a que éste solo funciona para máscaras en escalas de grises.

Función de optimización:

se utiliza la función del gradiente estocástico con un momentum de 0.9 y un learning rate de 0.1
Se utilizaron algunos pesos y bias de una red previamente entrenada para segmentación de objetos en RGB.

Especificaciones del hardware utilizado y del software elegido:

Debido a las operaciones utilizadas a través de la red neuronal, la cantidad de datos a utilizar y los procesos de entrenamiento, se recomienda ampliamente el uso de hardware especializado para el diseño, prueba e inclusive implementación de redes neuronales, en especial las redes neuronales convolucionales.

Se utilizó un servidor Intel Xeon con 2.30GHz con un hardware GPU de Tesla K80 utilizando la modalidad de notebooks Jupyter tanto para documentar como para ejecutar código (Google colab, 2017), debido al costo gratuito y a la facilidad de trabajo, siendo la manera más rápida y sencilla de trabajar que no requería el acceso a un hardware físico.

El lenguaje de programación utilizado fue python debido a la cantidad de módulos creados para cómputo científico y diseño de redes neuronales, además de ser el lenguaje que ejecuta el servidor.

Se eligió trabajar con la librería pytorch debido a las posibilidades que podía brindar, siendo ésta la que trabaja con la menor cantidad de librerías intermedias a diferencia de otras como keras, MXNet, CNTk, Caffe y Tensorflow (Deep Learning Frameworks Comparison, 2018), con un uso más generalizado en la academia y una curva de aprendizaje menor, posibilitando además la modificación de arquitecturas neuronales.

Especificaciones de Pytorch

Utiliza un tipo de datos llamados tensores que permiten trabajar con múltiples datos y trabajar con ellos utilizando el GPU, estos objetos permiten recordar las operaciones realizadas, generando la operación automática de diferenciación, utilizada para computar gradientes durante el aprendizaje (autograd) (Figura 37).

```
z = y * y * 3
out = z.mean()

print(z, out)
```

Out:

```
tensor([[27., 27.],
        [27., 27.]], grad_fn=<MulBackward0>) tensor(27., grad_fn=<MeanBackward0>)
```

Figura 37: Ejemplo de operaciones con gradiente automático, usando tensores.

Pytorch contiene un submódulo llamado NN quecontiene múltiples funciones implementadas para realizar capas neuronales, contiene por

ejemplo operaciones convolucionales, funciones de reducción y funciones de arquitecturas residuales (Figura 38).

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
       x = F.relu(self.conv1(x))
       return F.relu(self.conv2(x))
```

Figura 38: Ejemplo de módulo NN en pytorch.

Pytorch contiene además un módulo llamado Functionals que contiene funciones de activación, de pérdida e interpolaciones.
La programación de una red neuronal se realiza estructurando una clase de red, donde se especifiquen las partes que la contenga y como avanzan los datos a través de esta, utilizando la función inicializadora de python y una función con el nombre forward devolviendo al final un objeto tensor (Figura 38)(Understanding Net Class, 2017).

Pytorch contiene además una clase para facilitar la obtención de datos de un dataset, donde se puede especificar las transformaciones utilizadas para el aumento de datos, el número de lote tanto para el entrenamiento como para las pruebas (Data Loading and Processing Tutorial, 2017).

Por último para la parte de entrenamiento, se definen valores como learning rate, función de optimización y función de pérdida (Training a Classifier, 2017).

Dataset utilizado

Se investigó la existencia de algún dataset o recopilación de datos, el dataset tendría que tener información relevante de jitomates en un entorno agrícola, de preferencia en invernaderos, presentarse en un formato de imagen RGB, de preferencia que estuvieran etiquetadas las imágenes en 3 clases, exterior, vegetación o follaje y jitomates, tipo segmentaciones.

Debido a la nula existencia y la dificultad de encontrar un dataset con tales características, se realizó una recolección con la ténica webscrapping en diversas páginas que almacenan imágenes, con la finalidad de descargar una cantidad relevante, logrando descargar 1000 imágenes de jitomates en entorno de 3 fuentes diferentes (Figura 39).

Se eliminaron imágenes repetidas, se borraron imágenes donde aparecieran platillos hechos con jitomates y dibujos o sketches.

Se eligió utilizar la herramienta labelme para realizar el etiquetado debido a la baja curva de aprendizaje, la capacidad de trabajar con directorios de imágenes y la capacidad de exportar a formatos estandarizados utilizados para datasets., generando mapas de segmentación (Figura 40).

El programa de etiquetado exporta el resultado a archivos en cierto formato en particular en formato PASCAL VOC 2007, generando las carpetas “JPEGImages”, “SegmentationClass”, “SegmentationClassPNG” y “SegmentationClassVisualization” que contienen las imágenes usadas con el nombre en orden númerico, archivos de la matriz de clases, Imagenes de las clases y la mezcla de las imagenes originales con las etiquetas, respectivamente.

```
""""
  root/
    class_names.txt
    JPEGImages/
      0.jpg
      1.jpg
      ...
    SegmentationClass/
      0.npy
```

```
        1.npy
        ...
      SegmentationClassPNG/
        0.png
        1.png
        ...
      SegmentationClassVisualization/
        0.jpg
        1.jpg

"""
```

Se planteó desde un principio la necesidad de hacer aumento de datos tanto para generar más información para el entrenamiento como para generar una red que sea invariante a aspectos ambientales, variando el brillo, el contraste y el tono de las imágenes.

Además se reporta que el uso de aumento de datos permite tener mejoras en los tiempos de entrenamiento, en la precisión de los clasificadores y evita el overfitting (Wang, J., & Perez, L., 2017).

Se eligió las transformaciones en rotación, traslación, perspectiva y cambios en brilo, contraste y tono (Figura 41).

El dataset cuenta con un total de 700 imágenes etiquetadas, se dispuso de los datos en un repositorio público para futuras implementaciones.

Además para solucionar el problema debido a la sobreposición de jitomates, se decidió añadir un borde generado con dilatación y Canny para facilitar la tarea de instanciación de objetos separados (Figura 42).



Figura 39: Ejemplo de imagen del dataset

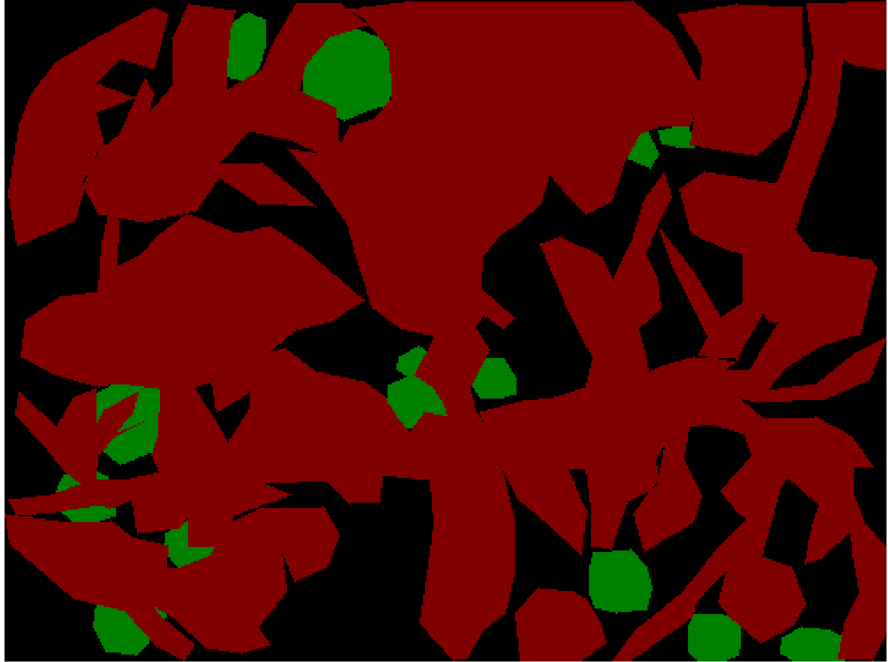


Figura 40: Ejemplo de imagen etiquetada de segmentación.



Figura 41: Aumento de datos.

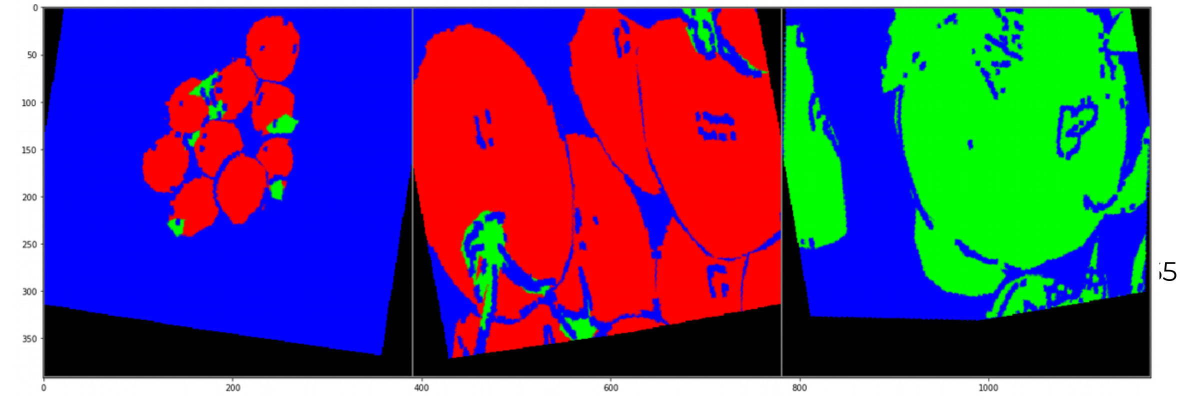


Figura 42: Etiquetas con borde añadido.

Resultados

Código para importar las librerías necesarias.

```
# operaciones de listado de archivos en el disco.

import os

from os import listdir

# operaciones de vision computacional (dilatación, umbralado, )
import cv2
import numpy as np

# necesario para replicar el mismo entrenamiento y transformación de datos
import random

# librerías necesarias para construir la red y para la transformación de imágenes
import torch
import torchvision
import torchvision.transforms as transf
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

# para darle el formato a las imágenes
import PIL
from PIL import ImageFile
from skimage import io

# para generar un dataset
from torch.utils.data import Dataset, DataLoader
```

Se crea la clase Dataset con las transformaciones para el aumento de datos, añadir el borde a las imágenes y disponer de ellas en lotes para el entrenamiento y la validación.

```
device = torch.device('cuda')
class TomatoDataset(Dataset):

  def __init__(self,root, transforms=None, train=False):
    self.root = root
```

```
    self.train = train

    # transformaciones geométricas (rotación, flip horizontal y vertical,
transformaciones de perspectiva)
    self.geomtransforms = transf.Compose([
      transf.RandomRotation(20, resample=PIL.Image.BILINEAR), # random rotation 20
degrees max.
      transf.RandomHorizontalFlip(),
      transf.RandomVerticalFlip(),
      transf.RandomAffine(degrees=(-10,10),translate=(0, 0.2)),
    ])

    # transformaciones de colores (brillo, contraste, saturación y matiz)
    self.colortransforms = transf.Compose([
      transf.ColorJitter(brightness=0.5, contrast=0.06, saturation=0.06, hue=0.05),
    ])

    jpegfolder = os.path.join(self.root,"JPEGImages/")
    segmentationfolder = os.path.join(self.root,"SegmentationClass/")

    # lista el nombre de los archivos y los ordena
    jpeglist = listdir(jpegfolder)
    segmlist = listdir(segmentationfolder)

    jpeglist.sort()
    segmlist.sort()

    # 80% entrenamiento - 20% prueba
    if self.train:

      # itera sobre la lista de archivos, lee el archivo, lo convierte a imagen y deja el 80%
de imagenes
      self.JPEGImages = [PIL.Image.fromarray(io.imread(jpegfolder+img)) for img in
jpeglist[:int(len(jpeglist)*0.8)]]
      self.edges = [io.imread(jpegfolder+img) for img in jpeglist[:int(len(jpeglist)*0.8)]]

      self.SegmentationClasss = [np.load(segmentationfolder+classes) for classes in
segmlist[:int(len(jpeglist)*0.8)]]
    else:
      # itera sobre la lista de archivos, lee el archivo, lo convierte a imagen y deja el 20%
de imagenes

      self.JPEGImages = [PIL.Image.fromarray(io.imread(jpegfolder+img)) for img in
jpeglist[int(len(jpeglist)*0.8):]]
      self.edges = [io.imread(jpegfolder+img) for img in jpeglist[int(len(jpeglist)*0.8):]]

      self.SegmentationClasss = [np.load(segmentationfolder+classes) for classes in
segmlist[int(len(jpeglist)*0.8):]]
```

```
        # utilizo la imagen original para obtener una máscara de los bordes, usando
operaciones de canny
        self.edges = [cv2.morphologyEx(img, cv2.MORPH_OPEN, np.ones((10,10))) for img in
self.edges]
        self.edges = [cv2.Canny(img, 200, 200) for img in self.edges]

        for n,img in enumerate(self.SegmentationClasss):

            # creo una imagen vacia del tamaño de la imagen
            blank = np.zeros((*img.shape,3),dtype=np.uint8)

            # asocio los canales R, G, B con las clases (R=tomate, G=follaje, B=fondo)
            blank[(img==2)] = np.array([255,0,0])
            blank[(img==1)] = np.array([0,255,0])
            [(img==0)] = np.array([0,0,255])

            # mezclo la máscara y el borde de color azul.
            mask = blank.copy()
            mask[np.where(self.edges[n]>0)] = np.array([0,0,255])

            img = PIL.Image.fromarray(mask)

            self.SegmentationClasss[n] = img

    def __getitem__(self,index):
        # utilizo la misma semilla de aleatoreidad para que se generen las mismas
transformaciones en etiquetas e imagenes
        seed = random.randint(0,2**32)
        random.seed(seed)

        # transformo la imagen número index, la normalizo y la convierto a un objeto tensor
        img = self.colortransforms(self.geomtransforms(self.JPEGImages[index]))
        img = transf.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))(transf.ToTensor()
(transf.Resize((572,572))(self.colortransforms(img))))
        random.seed(seed)

        if self.train:
            label = self.geomtransforms(
                transf.Resize((388,388))(self.SegmentationClasss[index]))
        else:

            label = transf.CenterCrop((388,388))(
                transf.Resize((572, 572))(
                    self.geomtransforms(
                        transf.Resize((388,388))(self.SegmentationClasss[index]))))

        label = transf.Normalize((0.5,), (0.5,))(transf.ToTensor()(label))
```

```
        return (img, label)

    def __len__(self):
        return len(self.SegmentationClasss)
```

Aqui se crea una clase Net para especificar como se mueve la imágen a través de la red neuronal.

```
# creamos la red neuronal
lossl =[]
device = torch.device('cuda')

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.pool = nn.MaxPool2d((2, 2), stride=(2,2))

        self.conv1 = nn.Conv2d(3, 64, 3)
        self.conv2 = nn.Conv2d(64, 128, 3)
        self.bn1 = nn.BatchNorm2d(64)

        self.conv3 = nn.Conv2d(128, 128, 3)
        self.conv4 = nn.Conv2d(128, 256, 3)
        self.bn2 = nn.BatchNorm2d(128)

        self.conv5 = nn.Conv2d(256, 256, 3)
        self.conv6 = nn.Conv2d(256, 512, 3)
        self.bn3 = nn.BatchNorm2d(256)

        self.conv7 = nn.Conv2d(512, 512, 3)
        self.conv8 = nn.Conv2d(512, 1024, 3)
        self.bn4 = nn.BatchNorm2d(512)

        self.conv9 = nn.Conv2d(1024, 1024, 3)
        self.conv10 = nn.Conv2d(1024, 2048, 3)
        self.bn5 = nn.BatchNorm2d(1024)
        self.bn6 = nn.BatchNorm2d(2048)

        self.conv11 = nn.Conv2d(128, 3, 1)

        self.upconv1 = nn.Conv2d(2048, 1024, 3)
        self.upconv2 = nn.Conv2d(1024, 512, 3)
        self.upconv3 = nn.Conv2d(512, 256, 3)
        self.upconv4 = nn.Conv2d(256, 128, 3)
```

```
def forward(self, x):


    x = x.to(device)

    # B 3 572 572
    x = nn.LeakyReLU()(
        self.bn1(self.conv1(x))
    )

    # B 64 570 570
    x = nn.LeakyReLU()(
        self.bn2(self.conv2(x))
    )


    crop4 = x[:,:,285-196:285+196, 285-196:285+196]
    # B 128 568 568
    x = self.pool(x)
    # B 128 284 284

    # B 128 284 284
    x = nn.LeakyReLU()(
        self.bn2(self.conv3(x))
    )

    # B 128 282 282
    x = nn.LeakyReLU()(
        self.bn3(self.conv4(x))
    )

    crop3 = x[:,:,140-100:140+100, 140-100:140+100]
    # B 256 280 280
    x = self.pool(x)
    # B 256 140 140

    # B 256 140 140
    x = nn.LeakyReLU()(
        self.bn3(self.conv5(x))
    )
    # B 256 138 138
    x = nn.LeakyReLU()(
        self.bn4(self.conv6(x))
    )

    # B 512 136 136
    crop2 = x[:,:,68-52:68+52, 68-52:68+52]
```

```
x = self.pool(x)
# B 512 68 68


# B 512 68 68
x = nn.LeakyReLU()(
   self.bn4(self.conv7(x))
)
# B 512 66 66
x = nn.LeakyReLU()(
   self.bn5(self.conv8(x))
)

# B 1024 64 64
crop1 = x[:,:,32-28:32+28, 32-28:32+28]
x = self.pool(x)
# B 1024 32 32


# B 1024 32 32
x = nn.LeakyReLU()(
   self.bn5(self.conv9(x))
)
# B 1024 30 30
x = nn.LeakyReLU()(
   self.bn6(self.conv10(x))
)
# B 2048 28 28
x5 = x


# expansor ----------------------------------------------------

# B 2048 28 28
x = F.interpolate(x5, scale_factor=2, mode='bilinear' , align_corners=True)
x = x[:,0:1024,:,:]

# B 1024 56 56  +  B 1024 56 56

up1 = torch.cat((crop1, x), 1)


x = nn.LeakyReLU()(
   self.bn6(up1)
)

# B 2048 56 56
x = nn.LeakyReLU()(
```

```
            self.bn5(self.upconv1(x))
        )
        # B 1024 54 54
        x4 = nn.LeakyReLU()(
            self.bn5(self.conv9(x))
        )

        # B 1024 52 52
        x = F.interpolate(x4, scale_factor=2, mode='bilinear', align_corners=True )
        x = x[:,0:512,:,:]

        # B 512 104 104  +  B 512 104 104

        up2 = torch.cat((crop2, x), 1)


        x = nn.LeakyReLU()(
            self.bn5(up2)
        )

        #B 1024 104 104
        x = nn.LeakyReLU()(
            self.bn4(self.upconv2(x))
        )
        # B 512 102 102
        x3 = nn.LeakyReLU()(
            self.bn4(self.conv7(x))
        )


        # B 512 100 100
        x = F.interpolate(x3, scale_factor=2, mode='bilinear' , align_corners=True)
        x = x[:,0:256,:,:]

        # B 256 200 200 + 256 200 200
        up3 = torch.cat((crop3, x), 1)

        x = nn.LeakyReLU()(
            self.bn4(up3)
        )

        # B 512 200 200
        x = nn.LeakyReLU()(
            self.bn3(self.upconv3(x))
        )
        # B 256 198 198
        x4 = nn.LeakyReLU()(
            self.bn3(self.conv5(x))
```

```
        )

        # B 256 196 196
        x = F.interpolate(x4, scale_factor=2, mode='bilinear' , align_corners=True)
        x = x[:,0:128,:,:]

        # B 128 196 196 + 128 196 196
        up4 = torch.cat((crop4, x), 1)

        x = nn.LeakyReLU()(
            self.bn3(up4)
        )

        # B 256 392 392
        x = nn.LeakyReLU()(
            self.bn2(self.upconv4(x))
        )
        # B 128 390 390
        x4 = nn.LeakyReLU()(
            self.bn2(self.conv3(x))
        )
        # B 128 388 388
        x = self.conv11(x4)
        x = nn.Softmax(dim=1)(x)

        return x

net = Net().to(device)
```

La red contiene los siguientes parámetros:

```
<bound method Module.parameters of Net(
  (pool): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1,
ceil_mode=False)
  (conv1): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1))
  (conv2): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1))
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (conv3): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1))
  (conv4): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1))
  (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (conv5): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1))
  (conv6): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1))
  (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
```

```
  (conv7): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1))
  (conv8): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1))
  (bn4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (conv9): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1))
  (conv10): Conv2d(1024, 2048, kernel_size=(3, 3), stride=(1, 1))
  (bn5): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (bn6): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (conv11): Conv2d(128, 3, kernel_size=(1, 1), stride=(1, 1))
  (upconv1): Conv2d(2048, 1024, kernel_size=(3, 3), stride=(1, 1))
  (upconv2): Conv2d(1024, 512, kernel_size=(3, 3), stride=(1, 1))
  (upconv3): Conv2d(512, 256, kernel_size=(3, 3), stride=(1, 1))
  (upconv4): Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1))
)>
```

Se especifican características de aprendizaje, así como la función de optimización utilizada y la función de pérdida.

```
learning_rate = 0.05
optimizer = optim.SGD(net.parameters(), lr=learning_rate, momentum=0.9)

criterion = nn.BCEWithLogitsLoss()
```

Se procede a iterar sobre los datos, pasarlos a través de la red y a utilizar el autogradiente para realizar el aprendizaje.

```
for epoch in range(10):  # loop over the dataset multiple times
running_loss = 0.0

for i, data in enumerate(trainloader, 0):
    # get the inputs; data is a list of [inputs, labels]
    inputs, labels = data

    # zero the parameter gradients
    optimizer.zero_grad()
    #scheduler.step()
```

```
        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs.cpu(), labels)

        loss.backward()
        optimizer.step()

        lossl.append(loss.item())

        # print statistics
        running_loss += loss.item()
        if i % 4 == 0:    # print every n minibatch
            print("{} loss: {} ".format(epoch + 1, running_loss ))
            running_loss = 0.0

print('Finished Training')
```

Para las operaciones de generación de centroides de imagen:

```
import cv2

from sklearn.cluster import KMeans
import numpy as np

segm = cv2.imread("mult1.png")
orig = cv2.imread("morig1.png")
img = segm[:,:,2]
_, img = cv2.threshold(img,127,255,cv2.THRESH_BINARY)

img = cv2.morphologyEx(img, cv2.MORPH_OPEN, np.ones((20,20)))

contours, hierarchy =
cv2.findContours(img,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
for c in contours:
   # calculate moments for each contour
   M = cv2.moments(c)

   # calculate x,y coordinate of center
   cX = int(M["m10"] / M["m00"])
   cY = int(M["m01"] / M["m00"])
   cv2.circle(orig, (cX, cY), 5, (255,255,255), -1)
   cv2.circle(img, (cX, cY), 5, (0,0,0), -1)
```

```
   cv2.putText(orig, f"{cX},{cY}", (cX - 15, cY -
15),cv2.FONT_HERSHEY_SIMPLEX, 0.4, (255, 255, 255), 2)

cv2.imwrite("resumb1.png", img)
cv2.imwrite("res1.jpg", orig)
```

Se realizaron pruebas de entrenamiento con el mismo número de épocas, probando con 2 modificaciones, con la red neuronal U-Net con entradas y salidas RGB, otra prueba agregando BatchNorm y otra agregando BatchNorm con LeakyRelu.

Al final graficando los datos de pérdida (Figura 43), se puede observar que la red neuronal original presenta un mayor tiempo necesario para que la función de pérdida se asentara, con una pérdida mayor que las otras tres modificaciones, la función de BatchNorm presenta un tiempo mucho menor de asentamiento que la red original y la modificación con Batchnorm y LeakyRelu aunque tiene un tiempo un poco mayor de asentamiento que la red con BatchNorm sigue siendo mejor que la red original, además de presentar los datos más bajos de pérdida, lo que demuestra que la red ha mejorado en el entrenamiento.

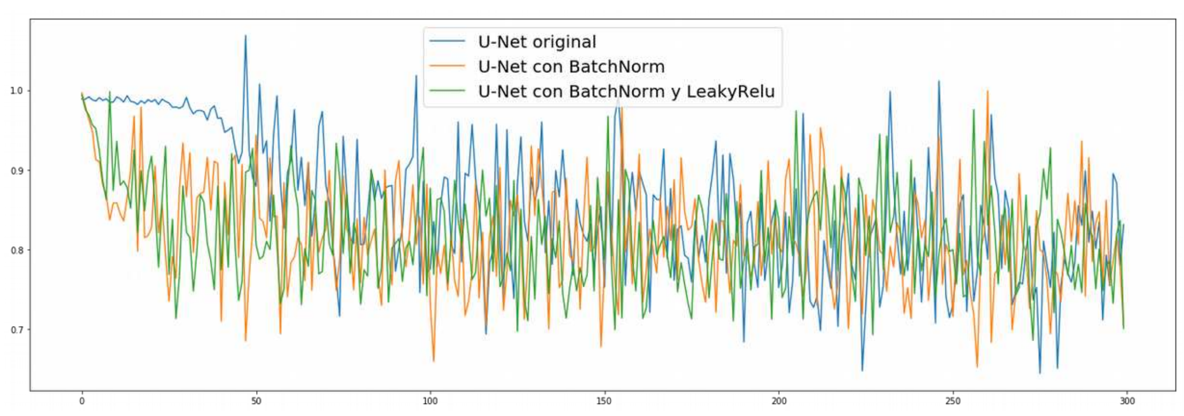


Figura 43: Comparativas de modificaciones a la red original

Se entrenó con 10 épocas con las 3 arquitecturas especificadas (U-Net original, con BatchNorm y con Batchorm y LeakyRelu) con aumento de datos, con un lote de 3 imágenes (debido a las dificultades de memoria en el servidor, se usaron 3)

Se obtuvieron las siguientes funciones de pérdida (Figura 44).

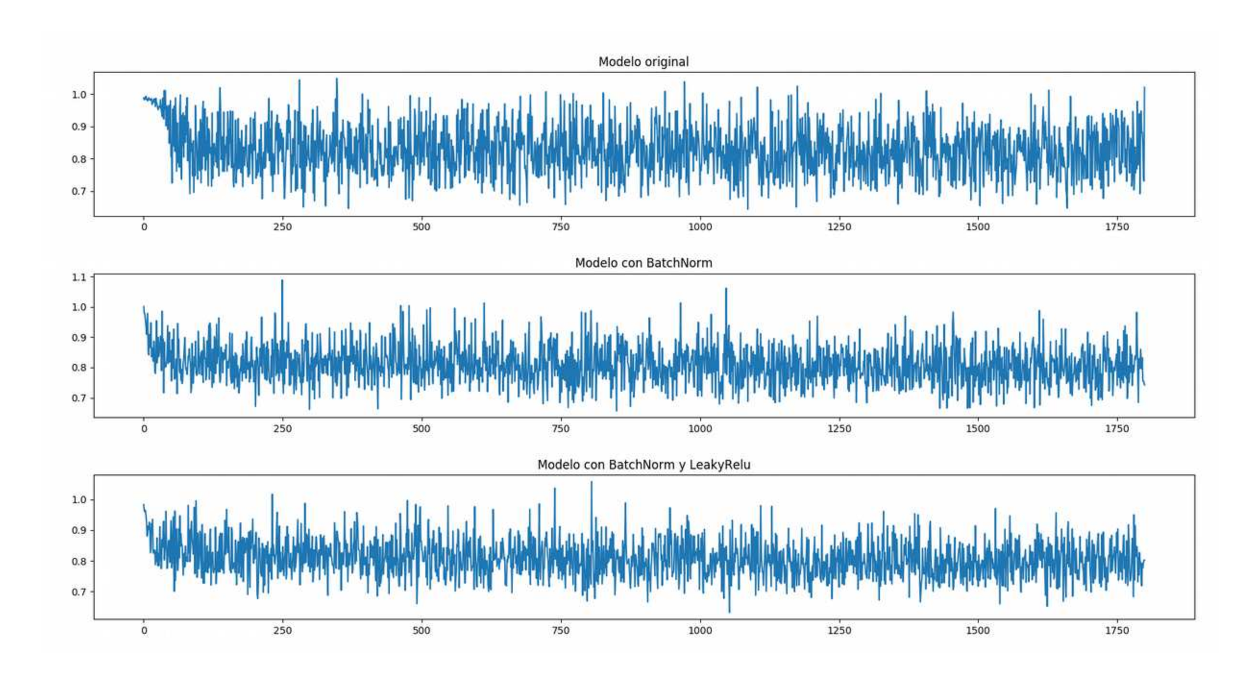


Figura 44: Funciones de pérdida.

Probando con un lote de pruebas (Figura 45) se pueden observar las modificaciones añadidas, observando los bordes añadidos en las subsecuentes etiquetas (Figura 46).

Se observan los resultados obtenidos, con las consideraciones de imagen centrada y etiquetado semántico (Figura 47, 48, 49).

El aprendizaje fue menor en la red original y de hecho se puede observar que aún no logra detectar el follaje.



Figura 45: Imagenes de prueba.

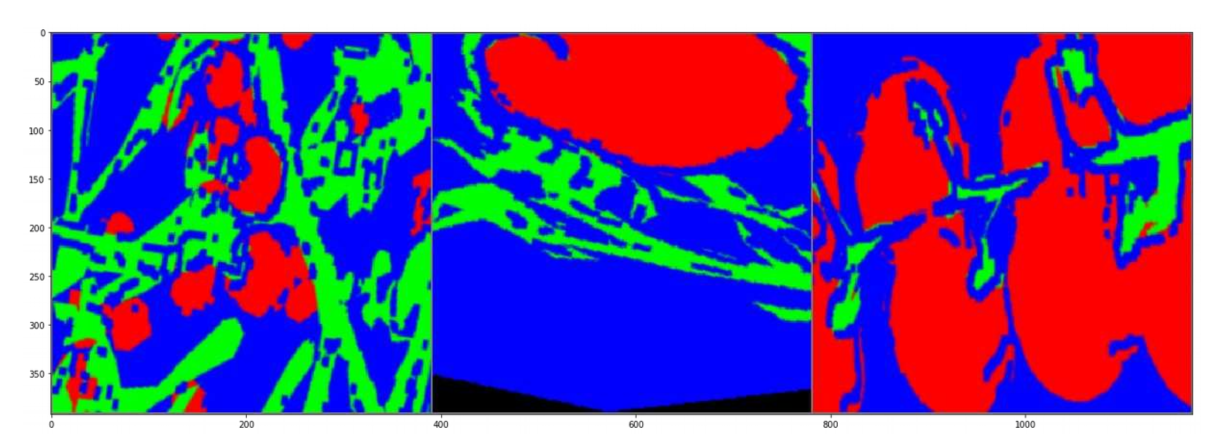


Figura 46: Etiquetado ground truth con borde añadido

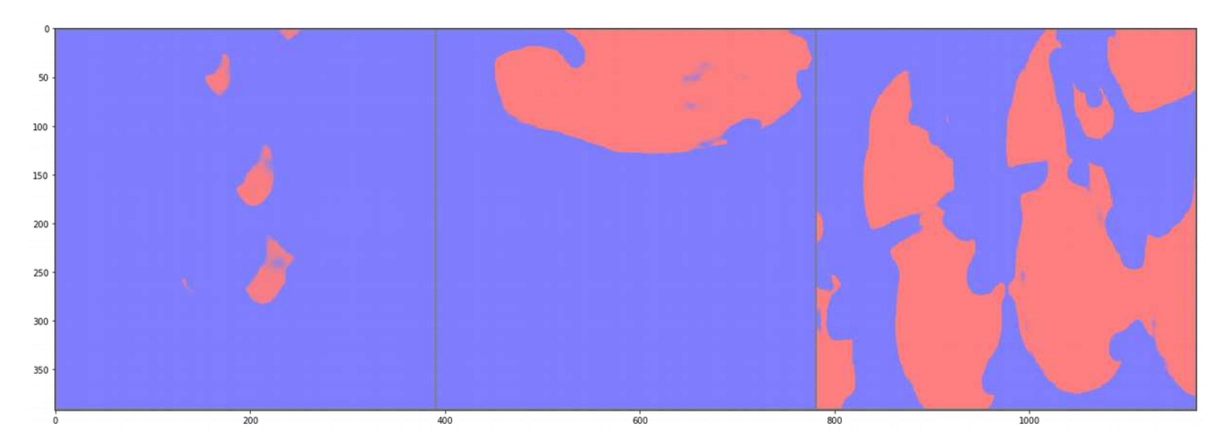


Figura 47: Resultado final red sin modificaciones.

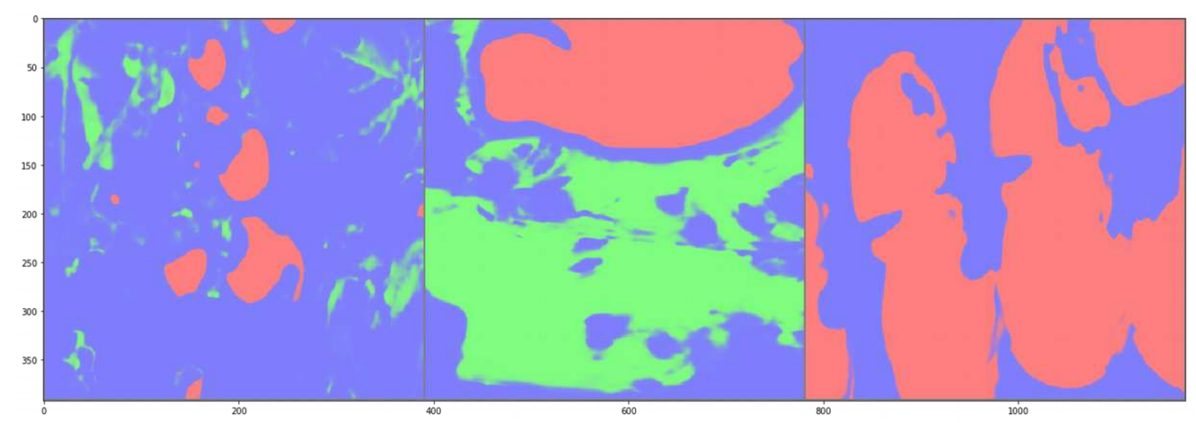


Figura 48: Resultados finales con BatchNorm.

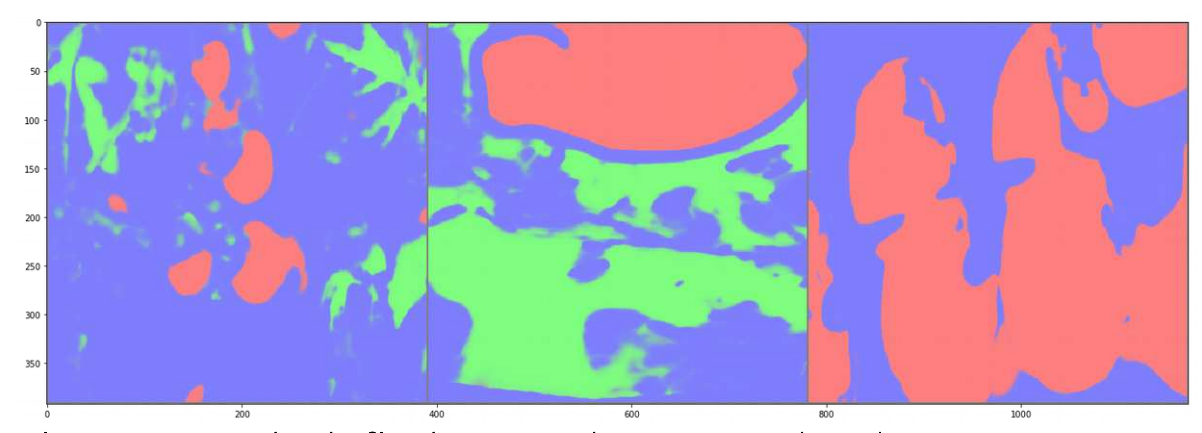


Figura 49: Resultado final con BatchNorm y LeakyRelu.



Figura 50: Segundo lote de imagenes de prueba.

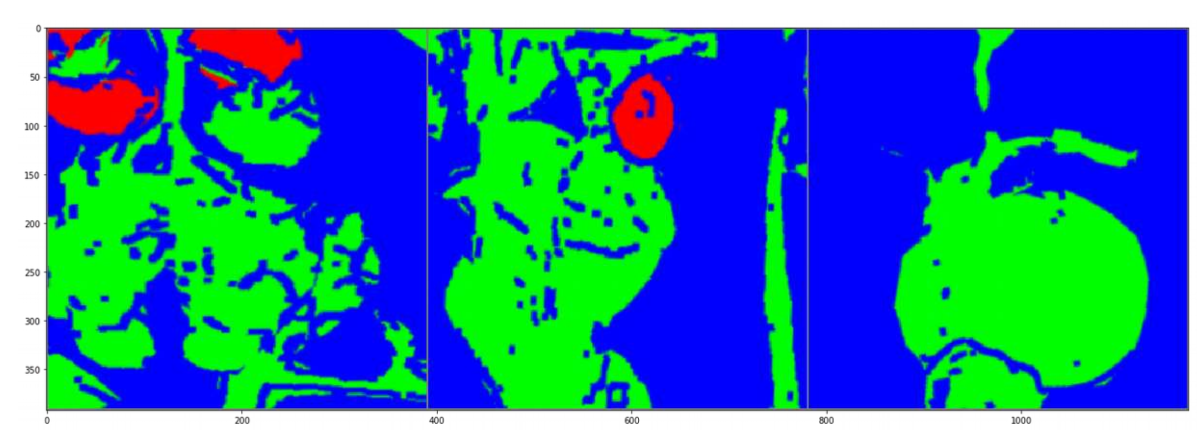


Figura 51: Etiquetado ground truth con borde añadido del segundo lote de imagenes de prueba.

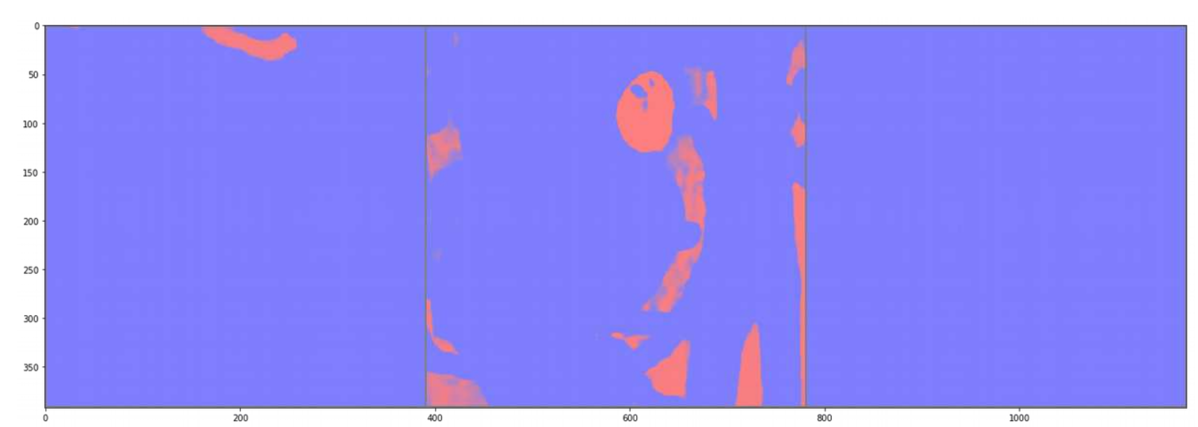


Figura 52: Resultado final del segundo lote de imagenes de prueba con la red sin modificaciones.

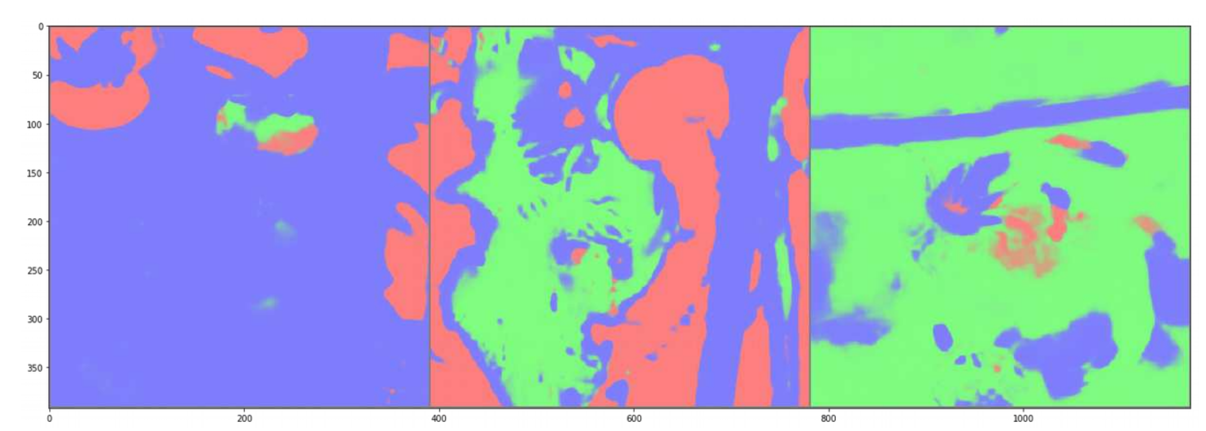


Figura 53: Resultado final del segundo lote de imágenes de prueba con la red con BatchNorm

Por último se realizó operaciones simples de umbralado y limpieza de la imagen con operaciones de visión computacional, abertura, utilizando solo el canal R del RGB.

Se utilizó además operaciones para encontrar contornos y momentum de una imagen, calculando los centroides de los blobs blancos.


Figura 54: Resultados de binarización, limpieza y cálculo de centroides.



Figura 55: Resultados finales con centroides

Conclusiones.

El diseño de los robots cosechadores requiere tener el conocimiento del entorno de trabajo, las capacidades técnicas y económicas con que se cuenta y algunas consideraciones previas de implementación, la investigación del estado del arte permitió elegir las opciones que se han utilizado previamente y dan un panorama de medición e intrumentación que se puede elegir para afrontar el problema.

Además, investigar características esenciales en el diseño e implementación de redes neuronales convolucionales, a través del estudio del estado del arte, contribuyó a la formulación de un modelo bien estructurado, con resultados más predecibles y de baja complejidad.

Es relevante el estudio a profundidad de las redes neuronales convolucionales para el funcionamiento óptimo y el menor consumo de recursos, en el caso de las redes clasificadoras, se busca que se tenga la mayor cantidad de características relacionada con información espacial menos difusa, esto sin comprometer el desempeño, en tiempos de entrenamiento y de prueba, se logró hacer más eficiente una red neuronal agregando parámetros de BatchNorm y LeakyRelu que significó mejoras a nivel de función de pérdida.

El uso de funciones no lineales permiten extrapolar información y generar mapas de características únicos que ayuden a generalizar la forma de las salidas de manera más precisa, puesto que se tiene mayor cantidad de información independiente entre sí, A pesar de ser esto algo conocido en las redes neuronales y más en particular las redes neuronales convolucionales, aún no se ha generado un panorama literario completo que explique la naturaleza de dichas funciones, más bien se permite el uso de funciones que a priori ha mostrado beneficios en algunas características, aquí se precisa el uso de funciones novedosas y algunas posibles evaluaciones, demostrando los beneficios de la implementación en nuestro caso de estudio.

Se logró tener un dataset de 800 imágenes de jitomates disponibles para redes neuronales segmentadoras, eligiendo la herramienta adecuada que permitió que el etiquetado fuera realizado de buena manera.

Además se realizó un ejemplo de detección de contornos y centroides de jitomates, obteniendo su posición en x y y, permitiendo en un futuro la posible implementación en robots cosechadores.

Recomendaciones y mejoras futuras.

Agregar una red instanciadora podría ayudar en la separación de jitomates de manera individual, además probar otras partes extractoras o técnicas profundas para la obtención de mapas de características más complejos ayudaría a tener una clasificación más precisa, menos dependiente de características de color y más de textura de los jitomates.

Aumentar el dataset y probar con nuevas funciones de transformación para los datos ayudaría a volver el problema más específico a jitomates y menos a colores rojos.

Debido a los tiempos, no se logró implementar los parámetros de desempeño del resultado de la red neuronal, ni los parámetros de desempeño a nivel visibilidad del robot, se espera lograr esto a futuro.

Competencias desarrolladas y/o aplicadas

- Destreza en el uso de software.
- Lectura de documentos de carácter científico o documental.
- Gestión del tiempo.
- Organizar información y realizar estrategias de aprendizaje.
- Tomar decisiones y resolución de problemas.
- Generar documentos bajo una metodología científica.
- Generación de datos novedosos.
- Normalizar los datos obtenidos.
- Síntesis y presentación de datos.
- Pruebas y correcciones de modelos computacionales.
- Capacidad de trabajo autónomo.
- Capacidad de trabajar bajo limitaciones.
- Capacidad de adaptarse a nuevas situaciones.
- Aplicar los conocimientos en la práctica.
- Trabajar en un horario laboral.
- Generar modelos computacionales a partir de formulaciones matemáticas.
- Consideraciones del uso de métricas.
- Investigación del estado del arte.
- Capacidad de agregar nuevas características a modelos previos.
- Capacidad de extrapolar modelos a tareas distintas a las originales.

Referencias bibliográficas y virtuales

Bachche, S. (2015). Deliberation on Design Strategies of Automatic Harvesting Systems: A Survey. *Robotics*, *4*(2), 194–222. https://doi.org/10.3390/robotics4020194

Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., … Murphy, K. (2016). Speed/accuracy trade-offs for modern convolutional object detectors. *arXiv:1611.10012 [cs]*. Recuperado de http://arxiv.org/abs/1611.10012

Juste, F., & Sevila, F. (1992). Citrus: an European project to study the robotic harvesting of oranges. *Rapport. Institut for Jordbrugsvidenskab (Denmark). no. 67.*

Bac, C. W., van Henten, E. J., Hemming, J., & Edan, Y. (2014). Harvesting Robots for High-value Crops: State-of-the-art Review and Challenges Ahead: Harvesting Robots for High-Value Crops: State-of-the-Art Review and Challenges Ahead. *Journal of Field Robotics*, *31*(6), 888–911. https://doi.org/10.1002/rob.21525

D. M. Bulanon, T. F. Burks, & V. Alchanatis. (2009). Fruit Visibility Analysis for Robotic Citrus Harvesting. *Transactions of the ASABE*, *52*(1), 277–283. https://doi.org/10.13031/2013.25933

Dale Whittaker, G. E. Miles, O. R. Mitchell, & L. D. Gaultney. (1987). Fruit Location in a Partially Occluded Image. *Transactions of the ASAE*, *30*(3), 0591–0596. https://doi.org/10.13031/2013.30444

Li, P., Lee, S., & Hsu, H.-Y. (2011). Review on fruit harvesting method for potential use of automatic fruit harvesting systems. *Procedia Engineering*, *23*, 351–366. https://doi.org/10.1016/j.proeng.2011.11.2514

C. E. Schertz and G. K. Brown. (1968). Basic Considerations in Mechanizing Citrus Harvest. *Transactions of the ASAE*, *11*(3), 0343–0346. https://doi.org/10.13031/2013.39405

Kawamura, N., NAMIKAWA, K., FUJIURA, T., & URA, M. (1984). Study on agricultural robot (part 1). *Journal of the Japanese Society of Agricultural Machinery*, *46*(3), 353-358.

d'Esnon, A. G. (1985). Robotic harvesting of apples. In *Agri-Mation 1, Chicago, Ill.(USA), 25-28 Feb 1985*. ASAE.

Sistler, F. (1987). Robotics and intelligent machines in agriculture. *IEEE Journal on Robotics and Automation*, *3*(1), 3–6. https://doi.org/10.1109/JRA.1987.1087074

Amaha, K.; Shono, H.; Takakura, T. A harvesting robot of cucumber fruits. Available online: http://agris.fao.org/agris-search/search.do?recordID=US9166423 (accessed on 15 June 2015).

Whitney, J. D., & Harrell, R. C. (1989). Status of citrus harvesting in Florida. *Journal of Agricultural Engineering Research*, *42*(4), 285–299. https://doi.org/10.1016/0021-8634(89)90031-0

Sittichareonchai, A., Sevila, F., Fatou, J. M., Constans, A., Brons, A., & Davenel, A. (1989). A robot to harvest grapes. In *ASAE Paper, No. 897084*.

Tillett, R.D. Initial development of a mechatronic mushroom harvester. In Proceedings of the International Conference on Mechatronics: Designing Intelligent Machines, Institution of Mechanical Engineers, Cambridge, UK, 1990; pp. 109–114

Sandini, G., Buemi, F., Massa, M., & Zucchini, M. (1990). Visually guided operations in green-houses. *IEEE International Workshop on Intelligent Robots and Systems, Towards a New Frontier of Applications*, 279–285. https://doi.org/10.1109/IROS.1990.262398

R. C. Harrell, P. D. Adsit, T. A. Pool, & R. Hoffman. (1990). THE FLORIDA ROBOTIC GROVE-LAB. *Transactions of the ASAE*, *33*(2), 0391–0399. https://doi.org/10.13031/2013.31342

Hayashi, U., & Ueda, Y. (1991). Orange harvesting robot. *Kubota" Co.. Sakai, Japan (Mirneo.)*, *280*.

T. A. Pool, & R. C. Harrell. (1991). An End-Effector for Robotic Removal of Citrus from the tree. *Transactions of the ASAE*, *34*(2), 0373–0378. https://doi.org/10.13031/2013.31671

Kassay, L. Hungarian robotic apple harvester. 1992 ASAE Annu. Meet. 1992, 92-7042, 1–14.

Benady, M.; Miles, G.E. Locating melons for robotic harvesting using structured light. ASAE Pap. No.: 92-7021, 1992

Tillett, N. D. (1993). Robotic Manipulators in Horticulture: A Review. *Journal of Agricultural Engineering Research*, *55*(2), 89–105. https://doi.org/10.1006/jaer.1993.1035

Reed, J. N., & Tillett, R. D. (1994). Initial experiments in robotic mushroom harvesting. *Mechatronics*, *4*(3), 265–279. https://doi.org/10.1016/0957-4158(94)90004-3

Grattoni, P., Cumani, A., Guiducci, A., & Pettiti, G. (1994, febrero 1). *Automatic harvesting of asparagus: an application of robot vision to agriculture* (W. J. Wolfe & W. H. Chun, Eds.). https://doi.org/10.1117/12.167496

Buemi, F. (1995). Agrobot: a robotic system for greenhouse operations. In *Proc. 4 Workshop on Robotics in Agriculture & the Food Industry, IARP, Toulouse, 1995*.

Edan, Y. (1995). Design of an autonomous agricultural robot. *Applied Intelligence*, *5*(1), 41–50. https://doi.org/10.1007/BF00872782

Kondo, N., Monta, M., & Fujiura, T. (1996). Fruit harvesting robots in Japan. *Advances in Space Research*, *18*(1–2), 181–184. https://doi.org/10.1016/0273-1177(95)00806-P

Arima, S., Kondo, N., & Nakamura, H. (1996). Development of robotic system for cucumber harvesting. *JARQ (Japan)*.

Mandow, A., Gomez-de-Gabriel, J. M., Martinez, J. L., Munoz, V. F., Ollero, A., & Garcia-Cerezo, A. (1996). The autonomous mobile robot AURORA for greenhouse operation. *IEEE Robotics & Automation Magazine*, *3*(4), 18–28. https://doi.org/10.1109/100.556479

Arndt, G., Rudziejewski, R., & Stewart, V. A. (1997). On the future of automated selective asparagus harvesting technology. *Computers and Electronics in Agriculture*, *16*(2), 137–145. https://doi.org/10.1016/S0168-1699(96)00033-6

Kondo, N., & Monta, M. (1999). Strawberry harvesting robots. *ASAE Pap*, (99-3071).

Reed, J. N., Miles, S. J., Butler, J., Baldwin, M., & Noble, R. (2001). AE—Automation and Emerging Technologies. *Journal of Agricultural Engineering Research*, *78*(1), 15–23. https://doi.org/10.1006/jaer.2000.0629

Brown, G. K. (2002). Mechanical harvesting systems for the Florida citrus juice industry. In *2002 ASAE Annual Meeting* (p. 1). American Society of Agricultural and Biological Engineers.

van Henten, E. J., Hemming, J., van Tuijl, B. A. J., Kornet, J. G., Meuleman, J., Bontsema, J., & van Os, E. A. (2002). An autonomous robot for harvesting cucumbers in greenhouses. *Autonomous Robots*, *13*(3), 241–258. https://doi.org/10.1023/A:1020568125418

Hayashi, S., Ganno, K., Ishii, Y., & Tanaka, I. (2002). Robotic Harvesting System for Eggplants. *Japan Agricultural Research Quarterly: JARQ*, *36*(3), 163–168. https://doi.org/10.6090/jarq.36.163

Cho, S. I. (2002). Development of a three-degrees-of-freedom robot for harvesting lettuce using machine vision and fuzzy logic control (S. I. Cho, S. J. Chang, Y. Y. Kim, & K. J. An, Trads.). *Biosystems engineering*, *v. 82*(2), 143–149. Recuperado de PubAg. (670025)

Van Henten, E. J., Van Tuijl, B. A. J., Hemming, J., Kornet, J. G., Bontsema, J., & Van Os, E. A. (2003). Field Test of an Autonomous Cucumber Picking Robot. *Biosystems Engineering*, *86*(3), 305–313. https://doi.org/10.1016/j.biosystemseng.2003.08.002

Seiichi Arima, Naoshi Kondo, & Mitsuji Monta. (2004). Strawberry Harvesting Robot on Table-top Culture. *2004, Ottawa, Canada August 1 - 4, 2004*. Presentado en 2004, Ottawa, Canada August 1 - 4, 2004. https://doi.org/10.13031/2013.16728

Burks, T., Villegas, F., Hannan, M., Flood, S., Sivaraman, B., Subramanian, V., & Sikes, J. (2005). Engineering and Horticultural Aspects of Robotic Fruit Harvesting: Opportunities and Constraints. *HortTechnology*, 79–87. https://doi.org/10.21273/HORTTECH.15.1.0079

Sanders, K. F. (2005). Orange Harvesting Systems Review. *Biosystems Engineering*, *90*(2), 115–125. https://doi.org/10.1016/j.biosystemseng.2004.10.006

Kitamura, S., & Oka, K. (2005). Recognition and cutting system of sweet pepper for picking robot in greenhouse horticulture. *IEEE International*

Conference Mechatronics and Automation, 2005, *4*, 1807–1812. https://doi.org/10.1109/ICMA.2005.1626834

Foglia, M. M., & Reina, G. (2006). Agricultural robot for radicchio harvesting. *Journal of Field Robotics*, *23*(6–7), 363–377. https://doi.org/10.1002/rob.20131

Belforte, G., Deboli, R., Gay, P., Piccarolo, P., & Ricauda Aimonino, D. (2006). Robot Design and Testing for Greenhouse Applications. *Biosystems Engineering*, *95*(3), 309–321. https://doi.org/10.1016/j.biosystemseng.2006.07.004

Naoshi Kondo, Shigemune Taniwaki, Koichi Tanihara, Kohki Yata, Mitsuji Monta, Mitsutaka Kurita, & Mitsuyoshi Tsutumi. (2007). n End-Effector and Manipulator Control for Tomato Cluster Harvesting Robot. *2007 Minneapolis, Minnesota, June 17-20, 2007*. Presentado en 2007 Minneapolis, Minnesota, June 17-20, 2007. https://doi.org/10.13031/2013.23427

Tanigaki, K., Fujiura, T., Akase, A., & Imagawa, J. (2008). Cherry-harvesting robot. *Computers and Electronics in Agriculture*, *63*(1), 65–72. https://doi.org/10.1016/j.compag.2008.01.018

Baeten, J., Donné, K., Boedrij, S., Beckers, W., & Claesen, E. (2008). Autonomous Fruit Picking Machine: A Robotic Apple Harvester. En C. Laugier & R. Siegwart (Eds.), *Field and Service Robotics* (Vol. 42, pp. 531–539). https://doi.org/10.1007/978-3-540-75404-6_51

Irie, N., Taguchi, N., Horie, T., & Ishimatsu, T. (2009). Asparagus harvesting robot coordinated with 3-D vision sensor. *2009 IEEE International Conference on Industrial Technology*, 1–6. https://doi.org/10.1109/ICIT.2009.4939556

Van Henten, E. J., Van't Slot, D. A., Hol, C. W. J., & Van Willigenburg, L. G. (2009). Optimal manipulator design for a cucumber harvesting robot. *Computers and Electronics in Agriculture*, *65*(2), 247–257. https://doi.org/10.1016/j.compag.2008.11.004

Scarfe, A. J., Flemmer, R. C., Bakker, H. H., & Flemmer, C. L. (2000). Development of an autonomous kiwifruit picking robot. *2009 4th*

International Conference on Autonomous Robots and Agents, 380–384. https://doi.org/10.1109/ICARA.2000.4804023

Chatzimichali, A. P., Georgilas, I. P., & Tourassis, V. D. (2009). Design of an advanced prototype robot for white asparagus harvesting. *2009 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, 887–892. https://doi.org/10.1109/AIM.2009.5229897

Aljanobi, A. A., Al-hamed, S. A., & Al-Suhaibani, S. A. (2010). A setup of mobile robotic unit for fruit harvesting. *19th International Workshop on Robotics in Alpe-Adria-Danube Region (RAAD 2010)*, 105–108. https://doi.org/10.1109/RAAD.2010.5524602

Hayashi, S., Shigematsu, K., Yamamoto, S., Kobayashi, K., Kohno, Y., Kamata, J., & Kurita, M. (2010). Evaluation of a strawberry-harvesting robot in a field test. *Biosystems Engineering*, *105*(2), 160–171. https://doi.org/10.1016/j.biosystemseng.2009.09.011

Kohan, A., Borghaee, A. M., Yazdi, M., Minaei, S., & Sheykhdavudi, M. J. (2011). Robotic harvesting of rosa damascena using stereoscopic machine vision. *World Applied Sciences Journal*, *12*(2), 231-237.

De-An, Z., Jidong, L., Wei, J., Ying, Z., & Yu, C. (2011). Design and control of an apple harvesting robot. *Biosystems Engineering*, *110*(2), 112–122. https://doi.org/10.1016/j.biosystemseng.2011.07.005

Li, P., Lee, S., & Hsu, H.-Y. (2011). Review on fruit harvesting method for potential use of automatic fruit harvesting systems. *Procedia Engineering*, *23*, 351–366. https://doi.org/10.1016/j.proeng.2011.11.2514

Feng, Q., Wang, X., Zheng, W., Qiu, Q., & Jiang, K. (2012). New strawberry harvesting robot for elevated-trough culture. *International Journal of Agricultural and Biological Engineering*, *5*(2), 1-8.

Li, W., & Kongzhi-Lilun-Zhuanye-Weiyuanhui (Eds.). (2012). *2012 31st Chinese Control Conference (CCC 2012): Hefei, China, 25 - 27 July 2012*. Piscataway, NJ: IEEE.

Zhenyu Yang, Wenqiang Zhang, Junxiong Zhang, Chunlong Zhang, Chao Ji, & Wei Li. (2013). Design and Experiment of Intelligent Monorail Cucumbers Harvester System. *2013 Kansas City, Missouri, July 21 - July 24, 2013*. Presentado en 2013 Kansas City, Missouri, July 21 - July 24, 2013. https://doi.org/10.13031/aim.20131608939

Hemming, J.; Bac, C.W.; Bart, A.J.; van Tuijl, B.A.J.; Barth, R.; Bontsema, J.; Pekkeriet, E. A robot for harvesting sweet-pepper in greenhouses. In Proceedings of the International Conference of Agricultural Engineering, Zurich, Switzerland, 6–10 July 2014;

Jiménez, A. R., Jain, A. K., Ceres, R., & Pons, J. L. (1999). Automatic fruit recognition: a survey and new results using Range/Attenuation images. *Pattern Recognition*, *32*(10), 1719–1736. https://doi.org/10.1016/S0031-3203(98)00170-8

E. A. Parrish, Jr., & A. K. Goksel. (1977). Pictorial Pattern Recognition Applied to Fruit Harvesting. *Transactions of the ASAE*, *20*(5), 0822–0827. https://doi.org/10.13031/2013.35657

Juste, F., & Sevila, F. (1992). Citrus: an European project to study the robotic harvesting of oranges. *Rapport. Institut for Jordbrugsvidenskab (Denmark). no. 67.*

Cardenas-Weber, M., Hetzroni, A., & Miles, G. E. (1991). Machine vision to locate melons and guide robotic harvesting. *Paper-American Society of Agricultural Engineers (USA).*

Dobrusin, Y., Edan, Y., Grinshpun, J., Peiper, U. M., & Hetzroni, A. (1992). Real-time image processing for robotic melon harvesting. *Paper-American Society of Agricultural Engineers (USA).*

Plá, F., Juste, F., & Ferri, F. (1993). Feature extraction of spherical objects in image analysis: an application to robotic citrus harvesting. *Computers and Electronics in Agriculture*, *8*(1), 57–72. https://doi.org/10.1016/0168-1699(93)90058-9

Pla, F. (1996). Recognition of Partial Circular Shapes from Segmented Contours. *Computer Vision and Image Understanding*, *63*(2), 334–343. https://doi.org/10.1006/cviu.1996.0023

Murali Regunathan, & Won Suk Lee. (2005). Citrus Fruit Identification and Size Determination Using Machine Vision and Ultrasonic Sensors. *2005 Tampa, FL July 17-20, 2005*. Presentado en 2005 Tampa, FL July 17-20, 2005. https://doi.org/10.13031/2013.19821

Chen, S. W., Shivakumar, S. S., Dcunha, S., Das, J., Okon, E., Qu, C., … Kumar, V. (2017). Counting Apples and Oranges With Deep Learning: A Data-Driven Approach. *IEEE Robotics and Automation Letters*, *2*(2), 781–788. https://doi.org/10.1109/LRA.2017.2651944

Zheng, Y.-Y., Kong, J.-L., Jin, X.-B., Wang, X.-Y., & Zuo, M. (2019). CropDeep: The Crop Vision Dataset for Deep-Learning-Based Classification and Detection in Precision Agriculture. *Sensors*, *19*(5), 1058. https://doi.org/10.3390/s19051058

Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. *arXiv:1505.04597 [cs]*. Recuperado de http://arxiv.org/abs/1505.04597

Grand, G. (1987). A self-propelled robot to pick apples. *American Society of Agricultural Engineers Paper*.

Levi, P., Falla, A., & Pappalardo, R. (1988). Image controlled robotics applied to citrus fruit harvesting. In *7th International Conference on Robot Vision and Sensory Controls, Zurich (Switzerland), 2-4 Feb 1988*. IFS Publications.

David C. Slaughter, & Roy C. Harrell. (1987). Color Vision in Robotic Fruit Harvesting. *Transactions of the ASAE*, *30*(4), 1144–1148. https://doi.org/10.13031/2013.30534

Peter W. Sites, & Michael J. Delwiche. (1988). Computer Vision to Locate Fruit on a Tree. *Transactions of the ASAE*, *31*(1), 0257–0265. https://doi.org/10.13031/2013.30697

Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv:1502.03167 [cs]*. Recuperado de http://arxiv.org/abs/1502.03167

D. M. Bulanon, T. F. Burks, & V. Alchanatis. (2009). Fruit Visibility Analysis for Robotic Citrus Harvesting. *Transactions of the ASABE*, *52*(1), 277–283. https://doi.org/10.13031/2013.25933

Van Henten, E. J., Hemming, J., van Tuijl, B. A. J., Kornet, J. G., Meuleman, J., Bontsema, J., & van Os, E. A. (2002). An Autonomous Robot for Harvesting Cucumbers in Greenhouses. *Autonomous Robots*, *13*(3), 241–258. https://doi.org/10.1023/A:1020568125418

Plebe, A., & Grasso, G. (2001). Localization of spherical fruits for robotic harvesting. *Machine Vision and Applications*, *13*(2), 70–79. https://doi.org/10.1007/PL00013271

Sa, I., Ge, Z., Dayoub, F., Upcroft, B., Perez, T., & McCool, C. (2016). DeepFruits: A Fruit Detection System Using Deep Neural Networks. *Sensors*, *16*(8), 1222. https://doi.org/10.3390/s16081222

Badrinarayanan, V., Kendall, A., & Cipolla, R. (2017). SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*(12), 2481–2495. https://doi.org/10.1109/TPAMI.2016.2644615

Chen, Y.-R., Chao, K., & Kim, M. S. (2002). Machine vision technology for agricultural applications. *Computers and Electronics in Agriculture*, *36*(2–3), 173–191. https://doi.org/10.1016/S0168-1699(02)00100-X

Feng, Q., Wang, X., Zheng, W., Qiu, Q., & Jiang, K. (2012). New strawberry harvesting robot for elevated-trough culture. *International Journal of Agricultural and Biological Engineering*, *5*(2), 1-8. https://doi.org/10.3965/j.ijabe.20120502.001

Labelme [software]. (2016). obtenido de https://github.com/wkentaro/labelme

Google Colab [software] (2017) obtenido de https://colab.research.google.com

“Deep Learning Frameworks Comparison” (6 de septiembre del 2018) obtenido de https://www.netguru.com/blog/deep-learning-frameworks-comparison

"Understanding Net Class" (mayo del 2017) obtenido de (https://discuss.pytorch.org/t/understanding-net-class/2557/6)

"Data Loading and Processing Tutorial" (2017) obtenido de https://pytorch.org/tutorials/beginner/data_loading_tutorial.html

"Training a Classifier" (2017) obtenido de https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html

Wang, J., & Perez, L. (2017). The effectiveness of data augmentation in image classification using deep learning. *Convolutional Neural Networks Vis. Recognit*.